JAVA CLASS AND OBJECTS

CLASS

- Class is a set of objects that shares common structures and common behaviors.
    - Here structure points to variables
    - Behavior points to methods
- Class is a user defined data type that includes a set of variables and methods.

Syntax

```
<modifier> class <class-name>
{
        // variable section
        // method section
}
```

Where,

- Modifier can be private, public, protected, default (friendly)
- Default modifier of a class is default modifier (friendly)

Example

```
public class JTest
{
        String name="ganesh";
        void disp()
        {
                // user code
        }
}
```

Declaring a class

- A class is declared by the use of the reserved keyword class
- Variables declared within a class are called as fields and functions declared inside a class are called as methods.

Characteristics of a class

- It must be declared using class keyword.
- It can've fields & methods
- It can've static & final methods
- It can be derived from another class (using extends keyword)
- It can be implemented from the interfaces. (implements keyword)
- A class can be preceded with an access modifier.

VARIABLES

- Variable is an identifier and it is a name given to the memory location
- Any number of variables can be added to the class.
- These variables that are declared inside the class are called Instance / Static variables.
- The declaration of instance variables doesn't occupy any space in memory. At the time of creation, they will occupy the memory in some extent.

Syntax

```
<access-modifier> <return-type> <name>; // declaration only
<access-modifier> <return-type> <name>=initial values; // definition of a variable
```

Example

```
// Heterogeneous variable declaration (Different Family)
        String name="Krishna";
        int id=99;
```

```
// Homogeneous variable declaration (Same Family)
        int a=35, b=95, c=90;
```

METHODS

- Any number of methods can be embedded in a class
- It supports four types of modifiers such as private, public, protected and default

Use

- It is mostly used to provide the implementation of variables(data).

Types

- Java supports two types of methods. They are
    1. Instance method
    2. Static method

Instance Method

- If a method is marked as any modifiers and without static modifier, then it is called as instance method
- It purely depends on object of a class. So it must be called using object

Static Method

- If a method is marked as static modifier, then it is called as static method
- It purely depends on class. So it must be called using class name not object
- It is an important to note that, it permits only static variable. It does not allow the use of instance variable

DIFFERENCE BETWEEN INSTANCE METHOD AND STATIC METHOD

S.N	INSTANCE METHOD	STATIC METHOD
1.	It purely depends on object	It purely depends on class
2.	It permits implementation of both instance variable and static variable	It is designed only for static members. So it allows only the implementation of static variable. It does not allow the instance variable within it
3.	It is marked as any modifiers except static	It must be marked with static modifier

Syntax

```
<modifier> <r.type> <u.name>
{
        // user code
}
```

Supported modifiers: (private, public, protected & friendly)

Example

```
public void disp()
{
        // user code
}
```

NOTE

- If you don't provide any modifier, then system will assign the friendly (default) as a default modifier to the method.

OBJECT CREATION

- An object is an instance of a class & it will be stored on heap memory
- It is a reference type (not value type)
- In java, an object is a block of memory that contains the storage space / memory to store all the instance fields.
- It is an important to note that, object occupies memory only for variables not methods. Because methods are independent of all objects of a particular class.
- Process of creating an object is called as instantiating an object (instantiation).
- Objects in java are created using the new operator.

NOTE

- In java, all reference types are created with help of new modifier.

Usage of new operator

- The new modifier is used to allocate the required memory automatically to the original object at the runtime & returns that original object as a reference to the reference variable

Objects creation

- Creating the objects of a class is either one step or two step process.

One Step Process

- Here we have to declare and define the object using new modifier at same time (combining both declaration and definition into one line)
- Optional

Two Step Process

- Optional
- First, we have to declare the objects. (declaration only)
- Second, we have to define / create the objects using the new modifier. This will create the actual objects. (definition)

Syntax

```
class-name ref-var=new constructor-name();  // declaration + definition

                    OR

class-name -name object-name;               // declaration only
object-name=new constructor-name ();        // definition or creation
```

Example (One Step Process)

- It is possible to combine declaration and definition in one line

```
Mobile nokia=new Mobile();          // declaration & definition
```

- The **Mobile()** is a default constructor of the class. We can create any number of objects of the same class.
- The above statement creates an original object & calls the default constructor (implicit constructor calling)
- This statement just creates an object and return that reference (address of original object) to object variable called 'nokia'.

(OR)

Example (Two Step Process)

```
Mobile nokia;                // object declaration
```

- We must allocate the memory to object before to use. It can be done by using **new** modifier.
- Compiler will provide error message, if null object is used for calling method

```
nokia=new Mobile();          // object definition
```

Pictorial representation

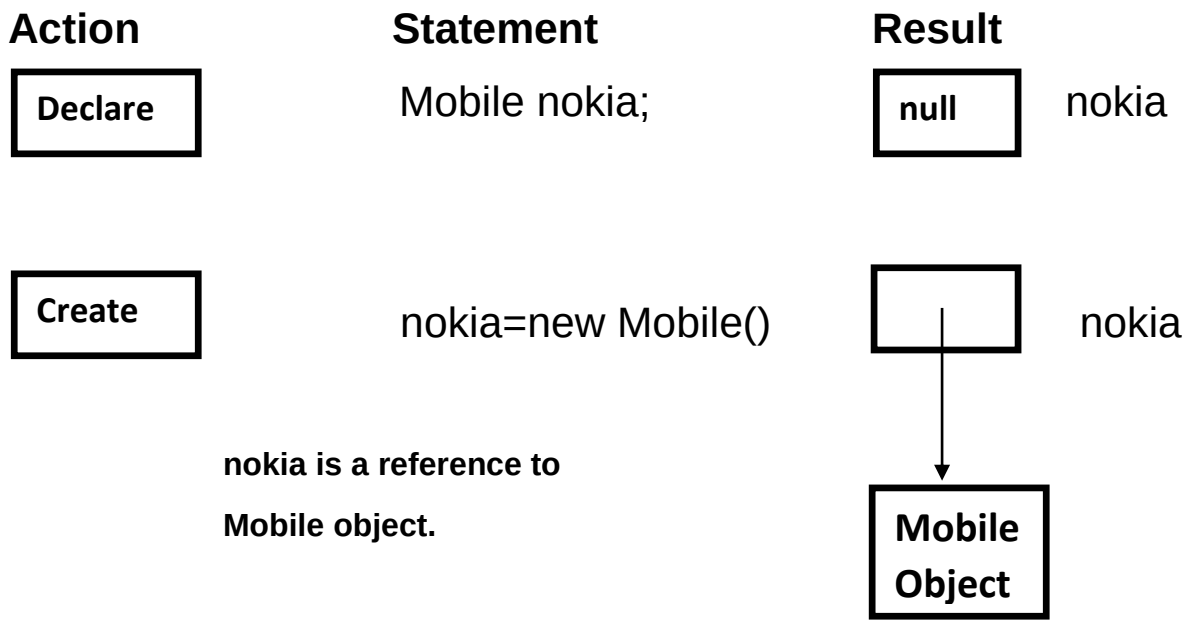


Figure: Creation of an object reference

NOTE

- It is an important to understand that, each object has its **own copy of the instance variables.** It means that, any changes to the values of an object have **no effect** on the variables of another object.

Multiple Objects

- It is possible to create multiple number of objects to a current class

Example

```
Mobile nokia=new Mobile();        // object 1
Mobile micro=new Mobile();        // object 2
…
etc                               // object n
```

Object Alias

- It is possible to assign one object reference directly (without using new modifier) to another object variable (object alias)

Example

```
Mobile nokia=new Mobile()      // create an original object & set its
                                  reference to nokia variable. Then
                                  later, nokia is called as nokia object
Mobile sony=nokia;             // assign the nokia object to sony
                                  object
```

Pictorial representation

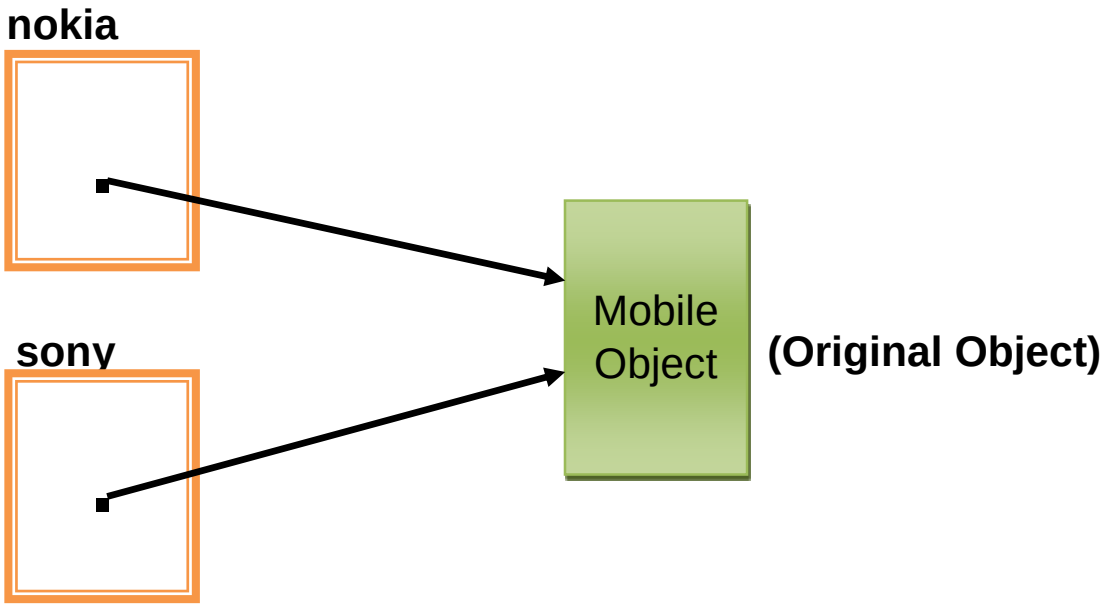


Figure: Assigning one object reference to another object reference

NOTE

- Here both “nokia” and “sony” points to same memory location / same object

DOT OPERATOR

Accessing Class Members

- Dot operator is used to access the variables and methods of a class
- It is used to access instance members (variables, methods) along with object
- It is also used to access static members (variables, methods) along with class name

Syntax (Instance Members)

```
object-name.variable-name;        // variable access
object-name.method-name();        // method access
```

Syntax (Static Members)

```
class-name.variable-name;     // static variable access
class-name.method-name();     // static method access
```

Example (Instance Members)

```
obj.name=”ragul”;              // Accessing instance variable
obj.disp(“good morning”);      // Accessing instance method
```

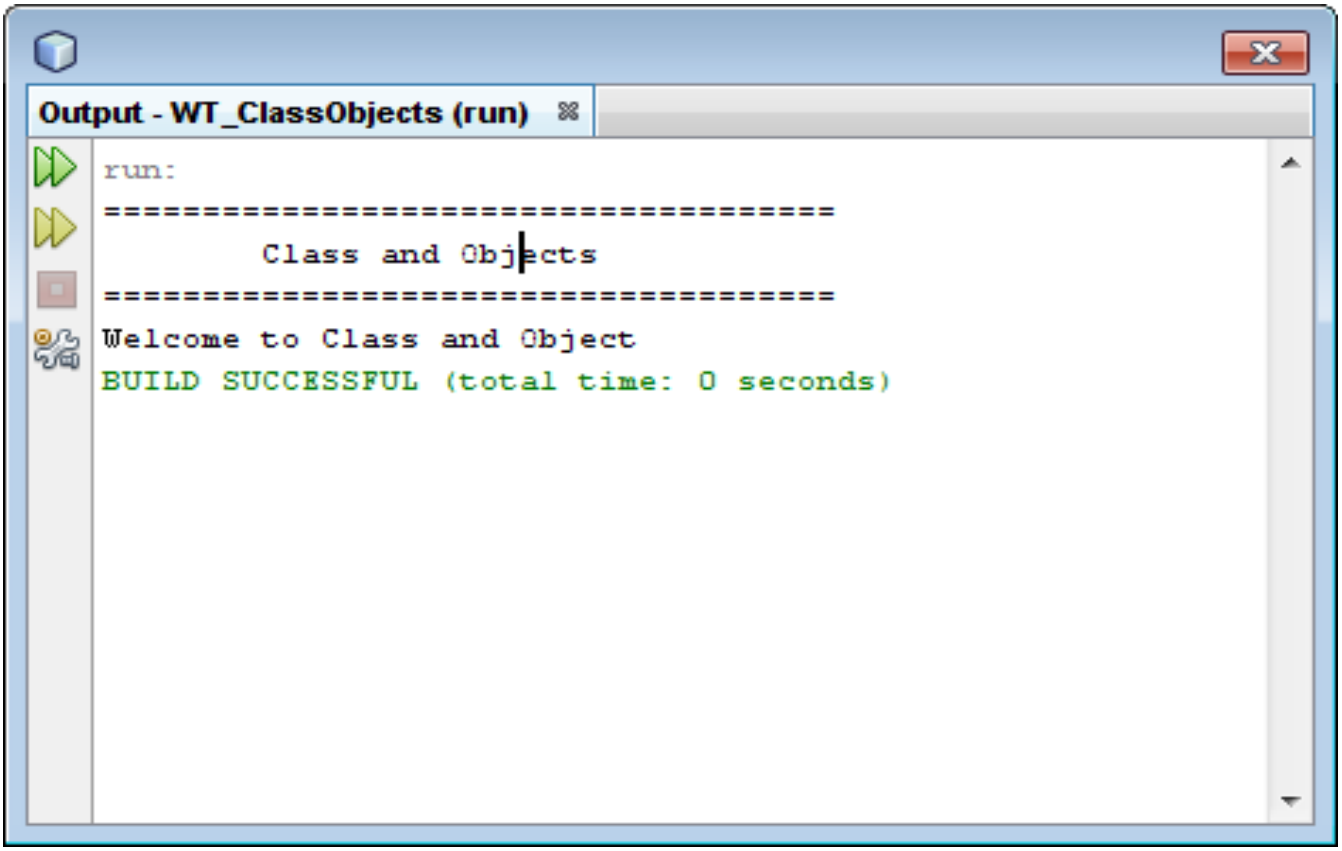
I. SIMPLE CLASS

(Simpleclass.java)

1. SOURCE CODE

```
// same class contains both implementation & calling
public class Simpleclass
{
// instance method
   void message()
   {
      System.out.println("Welcome to Class and Object");
   }
   public static void main(String[] args)
   {
// object creation using new modifier
      Simpleclass obj=new Simpleclass();
      System.out.println("=====================================");
      System.out.println("\tClass and Objects");
      System.out.println("=====================================");
// call the instance method
      obj.message();
   }
}
```

2. OUTPUT



II. CLASS WITH INSTANCE VARIABLES

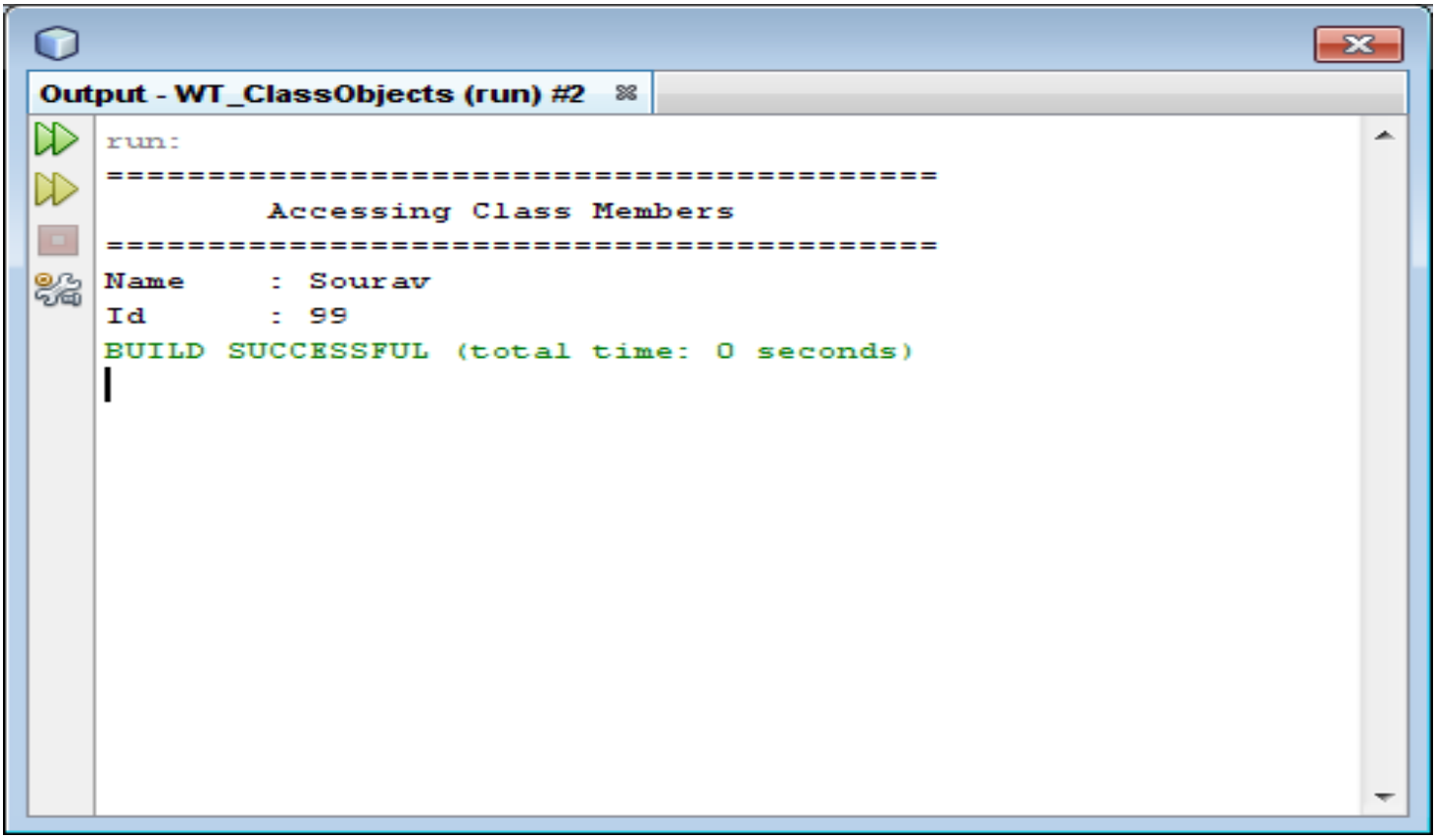
(Classwithdatamembers.java)

1. SOURCE CODE

```
// declaration class
class Student
{
// instance variable declarations
    String name;
    int id;
}
// calling class [Main Class]
public class Classwithdatamembers
{
// main method()
   public static void main(String[] args)
```

```
    {
        System.out.println("============================");
        System.out.println("\tAccessing Class Members");
        System.out.println("============================");
// object creation using new modifier
        Student obj=new Student();
// setting initial values class members in main() method
        obj.id=99;
        obj.name="Sourav";
// c-style printing
        System.out.printf("Name\t: %s\n",obj.name);
        System.out.printf("Id\t: %d\n",obj.id);
    }
}
```

2. OUTPUT



III. CLASS WITH INSTANCE METHODS

(Rectangle.java)

1. SOURCE CODE

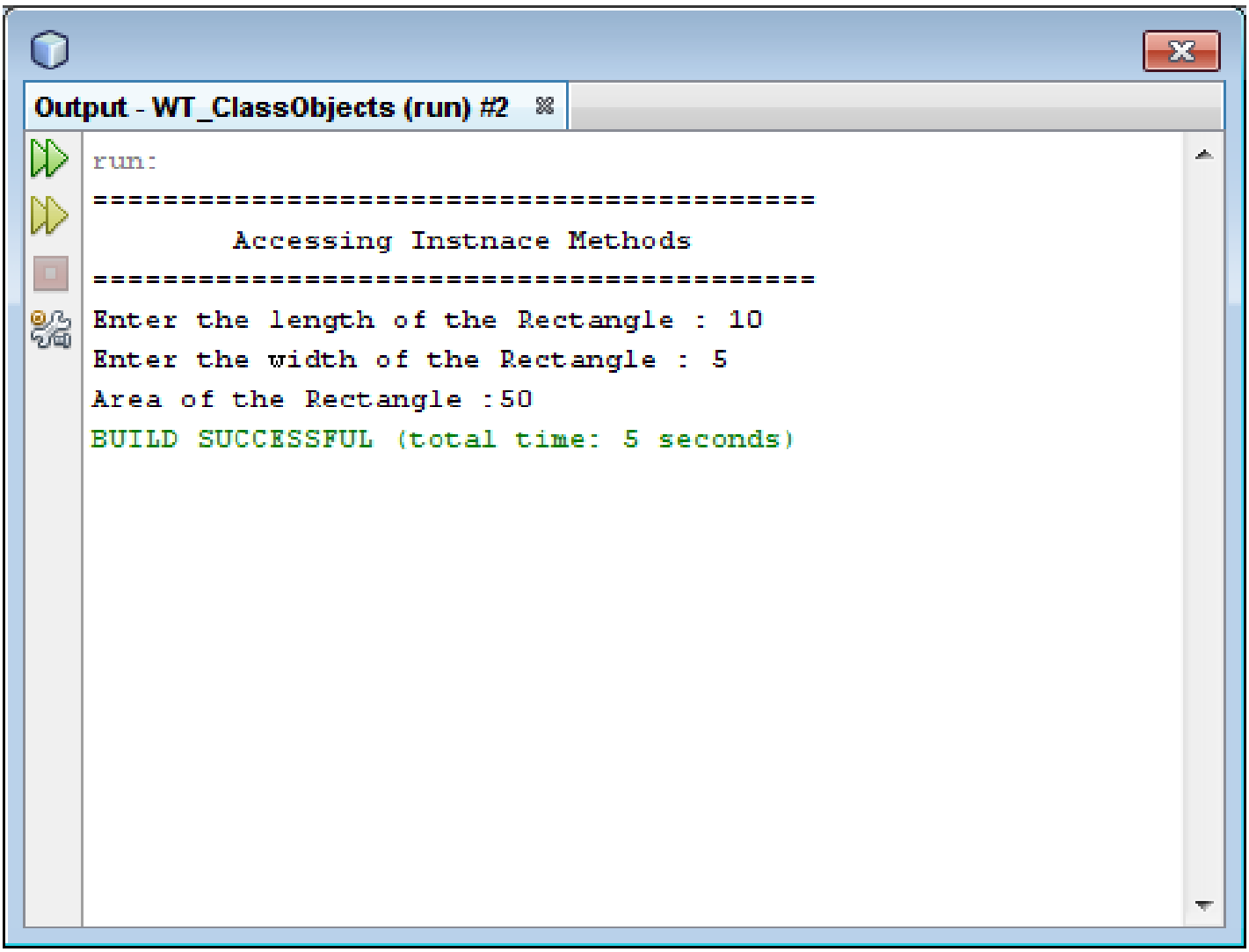
```
import java.io.*;                    // supports DataInputStream class
public class Rectangle
{
   int l, b, res;
// IO stream to read the keyboard input
   DataInputStream ds=new DataInputStream(System.in);
   void input()throws Exception
   {
      System.out.print("Enter the length of the Rectangle : ");
// read a line of text & convert it to int type using parsing technique
      l=Integer.parseInt(ds.readLine());
      System.out.print("Enter the width of the Rectangle : ");
      b=Integer.parseInt(ds.readLine());
   }
   void area()
   {
      res=l*b;
      System.out.printf("Area of the Rectangle :%d\n",res);
   }
   public static void main(String[] args)throws Exception
   {
      System.out.println("==========================");
      System.out.println("\tAccessing Instance Methods");
      System.out.println("==========================");
// object creation
      Rectangle rr=new Rectangle();
// calling the instance methods using object
      rr.input();
      rr.area();
```

```
    }
}
```

NOTE

- It is an important to note that, method must be marked as throws Exception or method must be defined within try catch block in case of using run time inputs.

2. OUTPUT



STATIC MEMBERS

STATIC VARIABLE

- If a variable is placed inside a class & outside of a method with static modifier, then it is called as static variable.
- Only one copy the static variable is shared by all the objects of a class. Because static variable purely belongs to a class.

Accessing

- To access the static variable, there is no need of creating the object of the class. They can be accessed with the use of class name itself.

```
class-name.variable-name;   // calling static variable
```

- If static members belongs to same class, then simply use the name of static variable (don't mention the class name).

```
variable-name;              // calling static variable
```

- Scope: till end of the class.

STATIC METHODS

- It is mainly used to call / access other static methods.
- It allows the static variable implementation (directly) & instance variable implementation. (using object)

Accessing

- Static methods are called by using class name.

```
class-name.static-method();     // calling static method
```

V. ACCESSING STATIC MEMBERS

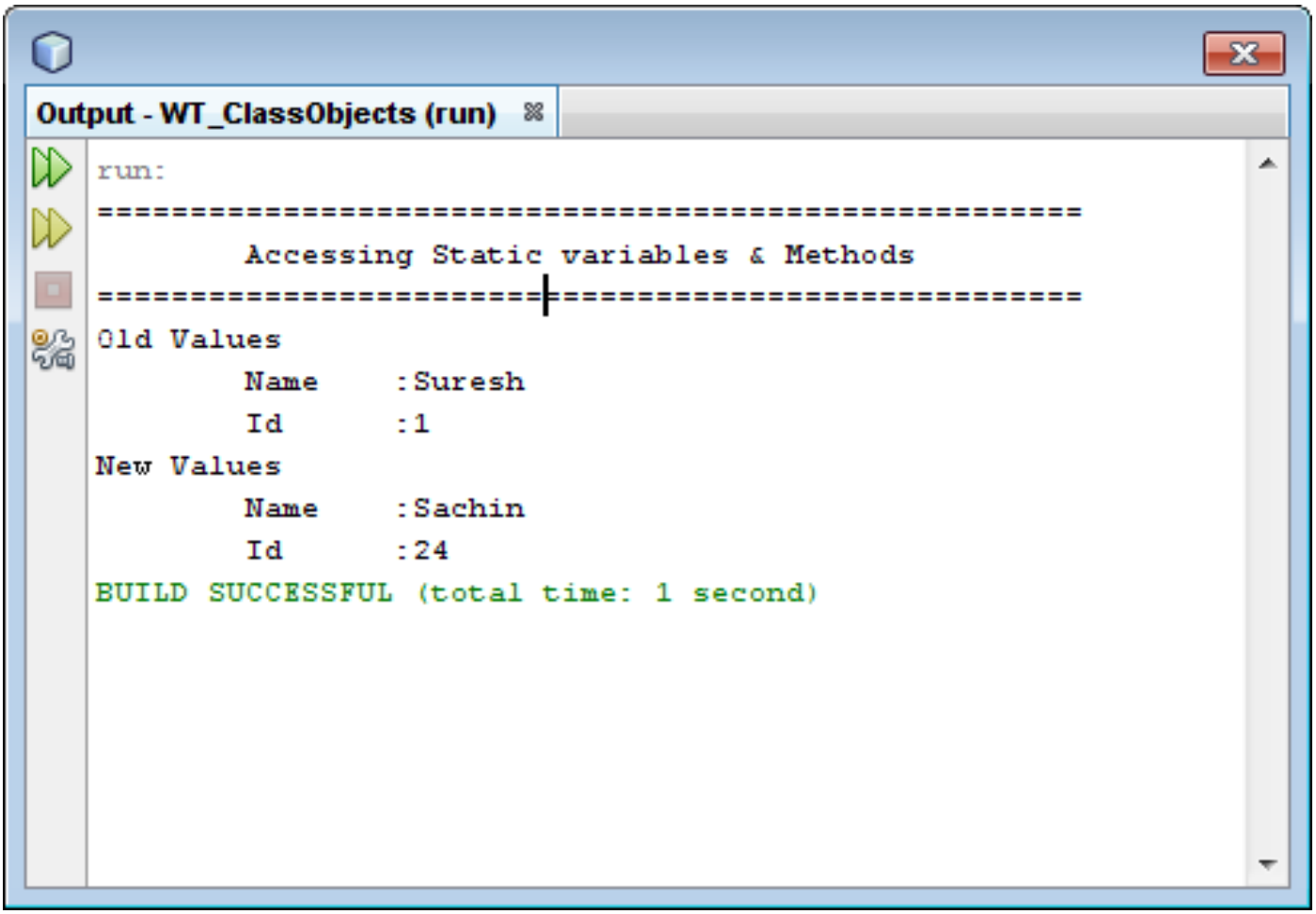
(StaticMembers.java)

1. SOURCE CODE

```
// main class
public class StaticMembers
{
// static members
    static int id;
    static String name;
// static method: (permit only static members)
    static void input(int i, String str)
    {
        id=i;
        name=str;
    }
// static method: permit only static members
    static void disp()
    {
        System.out.printf("Name \t:%s\n",name);
        System.out.printf("Id \t:%s\n",id);
    }
// main method
    public static void main(String[] args)
    {
        System.out.println("================================");
        System.out.println("\tAccessing Static variables & Methods");
        System.out.println("================================");
// accessing static variables
        id=01;
        name="Suresh";
        System.out.println("Old Values");
```

```
// calling static method
        disp();
// calling static method
        input(24,"Sachin");
        System.out.println("New Values");
// calling static method
        disp();
    }
}
```

2. OUTPUT



TYPES OF VARIABLES

- Java supports three types of variables. They are:
  1. Local variable
  2. Instance variable
  3. Static variable

I. LOCAL VARIABLE

- If a variable is placed inside the method, then it is called as local variable.
- It is shared by same method. It is not possible to call local variable to outside of other methods. Because its scope depends only on same method.
- It has no default values & no modifiers.
- It must be initialized before it is used in a programming.
- It does not depend on class & object. So no need to call local variable using object or class name

Scope

till end of a **same method**

Calling

using variable name

IX. USAGE OF LOCAL VARIABLE

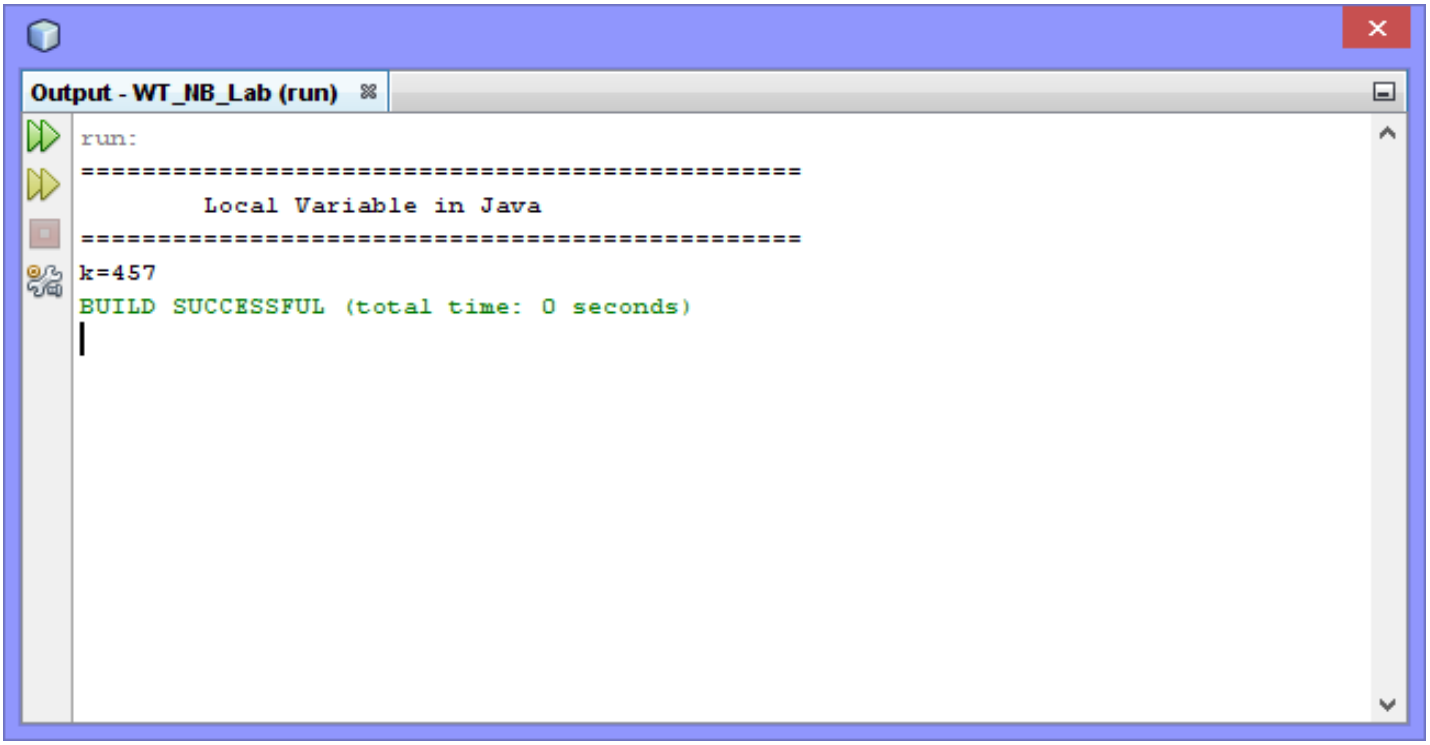
(Local_Variable.java)

1. SOURCE CODE

```
public class Local_Variable
{
// instance Method
    public void disp()
    {
        int k=457;
        System.out.println("k="+k);
    }
    public static void main(String[] args)
    {
 // object creation & calling default constructor using new modifier
        Local_Variable obj=new Local_Variable();
 // calling Instance method using object
        System.out.println("=======================");
        System.out.println("\tLocal Variable in Java");
        System.out.println("=======================");
        obj.disp();
    }
}
```

Definition of Local Variable

2. OUTPUT



II. INSTANCE VARIABLE

- If a variable is placed inside the class and outside of the methods, then it is called as Instance variable.
- This variable is used to store the information needed by multiple methods in objects (shared by multiple methods)
- Each copy of the instance variables is stored to each object of a particular class.
- It has default values and support modifiers.
- Initialization is optional.
- It purely depends on object. So object must be needed to call the instance members (variables, methods)

Scope

till end of a **same class**

Calling

using object name

X. USAGE OF INSTANCE VARIABLE

(Instance_Variable.java)

1. SOURCE CODE

Definition of Instance Variable

```
public class Instance_Variable
{
// instance variable declaration
// i is accessed by multiple instance methods in class.i.e{m1(),m2(),m3()}
    public int i=10;
// instance method 1: m1()
    public void m1()
    {
        System.out.println("i="+i);
        i*=10;
    }
// instance method 1: m2()
    public void m2()
    {
        System.out.println("i="+i);
        i*=10;
```

```
    }
// instance method 1: m3()
    public void m3()
    {
        System.out.println("i="+i);
    }
// main method
    public static void main(String[] args)
    {
// object creation using new operator
        Instance_Variable obj=new Instance_Variable();
// calling methods
        System.out.println("============================");
        System.out.println("\tInstance Variable in Java");
        System.out.println("============================");
        obj.m1();
        obj.m2();
        obj.m3();
    }
}
```

2. OUTPUT

```
Output - WT_NB_Lab (run)
run:
================================================
        Instance Variable in Java
================================================
i=10
i=100
i=1000
BUILD SUCCESSFUL (total time: 0 seconds)
```

III. STATIC VARIABLE (CLASS VARIABLE)

- If a variable is placed inside the class and outside of a method with static modifier, then it is called as static variable.
- Only one copy of the static variable is shared by all the objects of a class.
- Like instance variable, it is accessed by all other methods of same class
- It has default values & support only static modifier.
- Initialization is optional.

- It purely depends on class. So class must be needed to call the static members (static variable and methods)
- It can be accessed without creating objects.

Scope

till end of a **same class**

Calling

using class name

XI. USAGE OF STATIC VARIABLE

(Static_Variable.java)

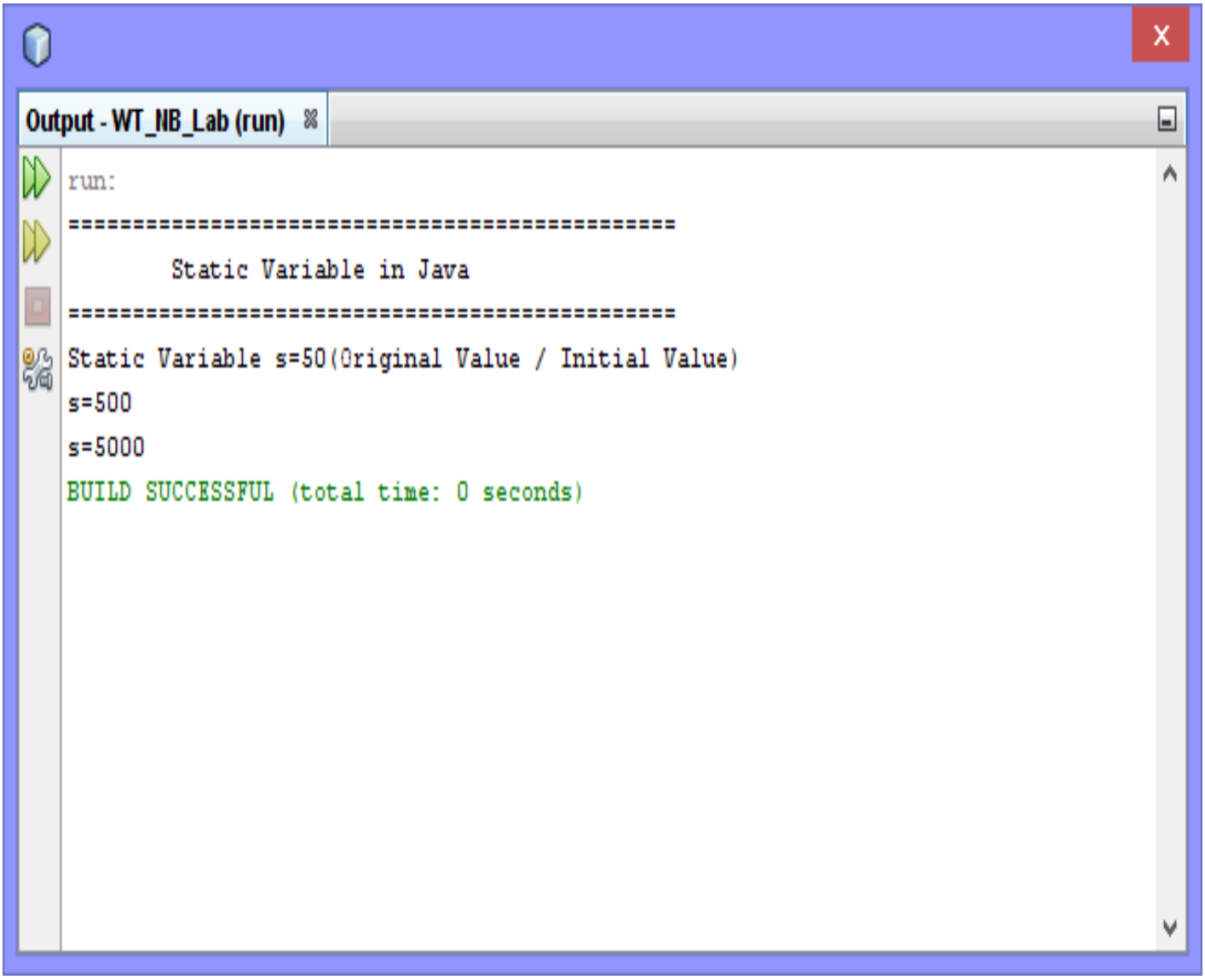
1. SOURCE CODE

```
public class Static_Variable
{
// static variable definition
    static int s=50;
// static method 1: m1()
    static void m1()
    {
        s*=10;
        System.out.println("s="+s);
    }
// static method 1: m2()
    static void m2()
    {
        s*=10;
        System.out.println("s="+s);
```

Definition of Static Variable

```
    }
// main method
    public static void main(String[] args)
    {
        System.out.println("======================");
        System.out.println("\tStatic Variable in Java");
        System.out.println("======================");
// calling static data & methods using class name
        System.out.println("Static Variable s="+Static_Variable.s+"(Original Value / Initial Value)");
        Static_Variable.m1();
        Static_Variable.m2();
    }
}
```

2. OUTPUT


Output - WT_NB_Lab (run)
run:
==========
Static Variable in Java
==========
Static Variable s=50(Original Value / Initial Value)
s=500
s=5000
BUILD SUCCESSFUL (total time: 0 seconds)

XII. DIFFERENCE BETWEEN INSTANCE & STATIC VARIABLES

(Diff_Ins_Stat.java)

1. SOURCE CODE

```
public class Diff_Ins_Stat
{
 // instance variable definition
        public int i=500;
// static variable definition
         static int s=100;
// constructor: used to update the both variables for every object creation
    public Diff_Ins_Stat()
    {
        s+=100;
         i+=100;
    }
// display the value of instance and static variables
    public void print()
    {
        System.out.println("\tInstance Data\t: "+i);
        System.out.println("\tStatic Data\t: "+s);
    }
// main method
    public static void main(String[] args)
    {
// object declaratons
        Diff_Ins_Stat obj1,obj2,obj3;
// create 1st object & call its default constructor
```

```
        obj1=new Diff_Ins_Stat();
// create 2rd object & call its default constructor
        obj2=new Diff_Ins_Stat();
// create 3rd object & call its default constructor
        obj3=new Diff_Ins_Stat();
        System.out.println("==================================");
        System.out.println("\tDiff b/w Instance Variable & Static Variable");
        System.out.println("==================================");
// calling all Instance methods
        System.out.println("Object 1:");
        obj1.print();
        System.out.println("Object 2:");
        obj2.print();
        System.out.println("Object 3:");
        obj3.print();
    }
}
```

2. OUTPUT

Output - WT_NB_Lab (run)

```
run:
==============================================================
        Diff b/w Instance Variable & Static Variable
==============================================================
Object 1:
        Instance Data   : 600
        Static Data     : 400
Object 2:
        Instance Data   : 600
        Static Data     : 400
Object 3:
        Instance Data   : 600
        Static Data     : 400
BUILD SUCCESSFUL (total time: 0 seconds)
```

COMPARISON OF LOCAL, INSTANCE AND STATIC VARIABLES

S. N	Local Variable	Instance Variable	Static Variable
1.	It is placed **inside a method.**	It is placed **inside a class.**	It is placed **inside a class with static** modifier.
2.	**Scope**: till end of the same method	**Scope**: till end of the same class	**Scope**: till end of the same class
3.	It has no default values	It has default values. So **no need to give initial values** during the variable creation.	It has default values. So **no need to give initial values** during the variable creation.
4.	It does not support modifiers	It supports modifiers	It supports **only static** modifier
5.	Initialization must be given**,** before it is used in the code.	Initialization is optional	Initialization is optional
6.	It does not depend on class and object.	It purely depends on **object.**	It purely depends on **class.**
7.	Calling It can be accessed by **variable name**.	Calling It can be accessed by **object name**.	Calling It can be accessed by **class name**.
8.	Example int k=457;	Example public String name;	Example static float a=45.74f;

JAVA BLOCKS

- Java supports three different types of blocks. They are
  1. Local block
  2. Instance block
  3. Static block

LOCAL BLOCK

- If a block is placed inside of a method, then it is called as local block
- It is identified by curly braces {}
- Unlike method, it does not support modifier, return type, etc, …
- It does not depend on class or object. So no need to call local block using object or class

Example

```
class LocalBlock
{
// instance method
        void disp()
        {
// local block inside a method
                {
                        System.out.println("Local Block is calling …");
                }
        }
// main method
        public static void main(String[] arg)
        {
// object creation
                LocalBlock obj=new LocalBlock();
// calling instance method using object
                obj.disp()
        }
}
```

OUTPUT

Local Block is calling …

INSTANCE INITIALIZATION BLOCKS (INSTANCE BLOCKS)

- If a block is placed inside a class and outside of all methods without any modifier, then it is called as instance block (initialization block)
- It has no name. It is identified by curly braes {}
- It is alternative to constructors / methods.
- It is mainly used to initialize the data members (instance variable & static variable) of the class (initialization purpose)
- It will be called every time automatically, when an object of a class is created.
- It is important to note that, it is called before the default constructor.
- All instance blocks are executed in sequential order

Calling

- It will be called automatically, when an object is created (before the default constructor).

Syntax

```
{
        // initialization code
}
```

Example

```
{
        System.out.println("Calling local block");
}
```

VI. INSTANCE BLOCKS / INITIALIZATION BLOCKS

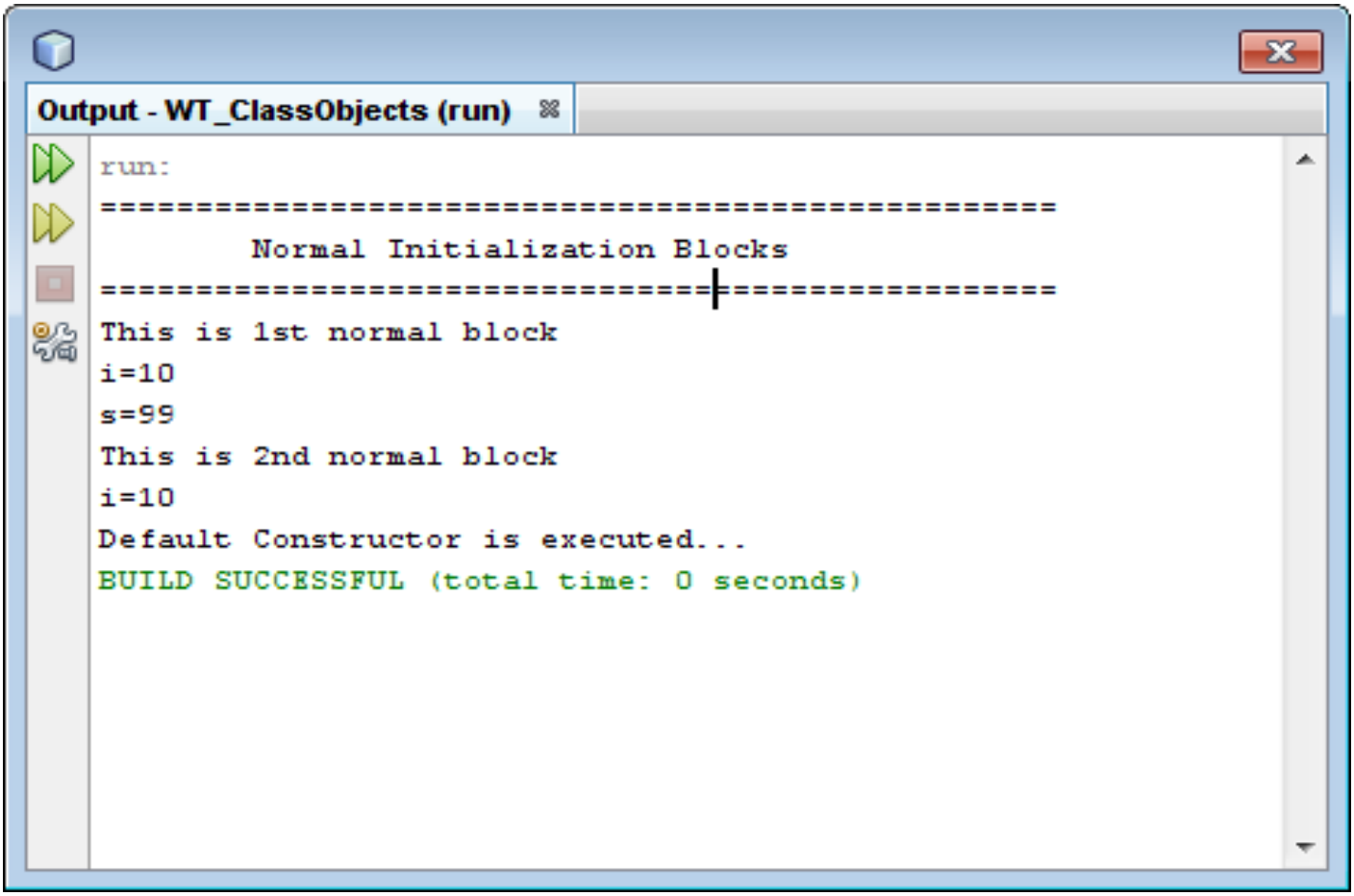
(InsBlocks.java)

1. SOURCE CODE

```
// main class
public class InsBlocks
{
// variables definition
   int i=10;
   static int s=99;
// normal Initialization Block 1 (instance block 1)
   {
      System.out.println("This is 1st normal block");
      System.out.println("i="+i);
      System.out.println("s="+s);
   }
// normal Initialization Block 2 (instance block 2)
   {
      System.out.println("This is 2nd normal block");
      System.out.println("i="+i);
   }
// default Constructor
   InsBlocks()
   {
      System.out.println("Default Constructor is executed...");
   }
```

```
// main Method
    public static void main(String a[])
    {
        System.out.println("==========================");
        System.out.println("\tNormal Initialization Blocks");
        System.out.println("============================");
        InsBlocks obj=new InsBlocks();
    }
}
```

2. OUTPUT



STATIC BLOCKS (STATIC INITIALIZATION BLOCKS)

- It is alternative to constructor and method.
- If a block is placed inside a class and outside of all the methods with static modifier, that is called static block
- Static Initialization blocks run only once when the class is first loaded.
- All static initialization blocks will be executed in the order they appear (executed in sequential order)
- A static initialization block is a normal block of code enclosed in braces, { }, and proceeded by the static keyword.
- It is used to initialize the static data members (instance variable-using object) of the class.

Calling

- It is called, when the class is loaded (before creating an object).

Syntax

```
static
{
        // initialization code
        // calling other static methods
}
```

VII. STATIC BLOCKS-1

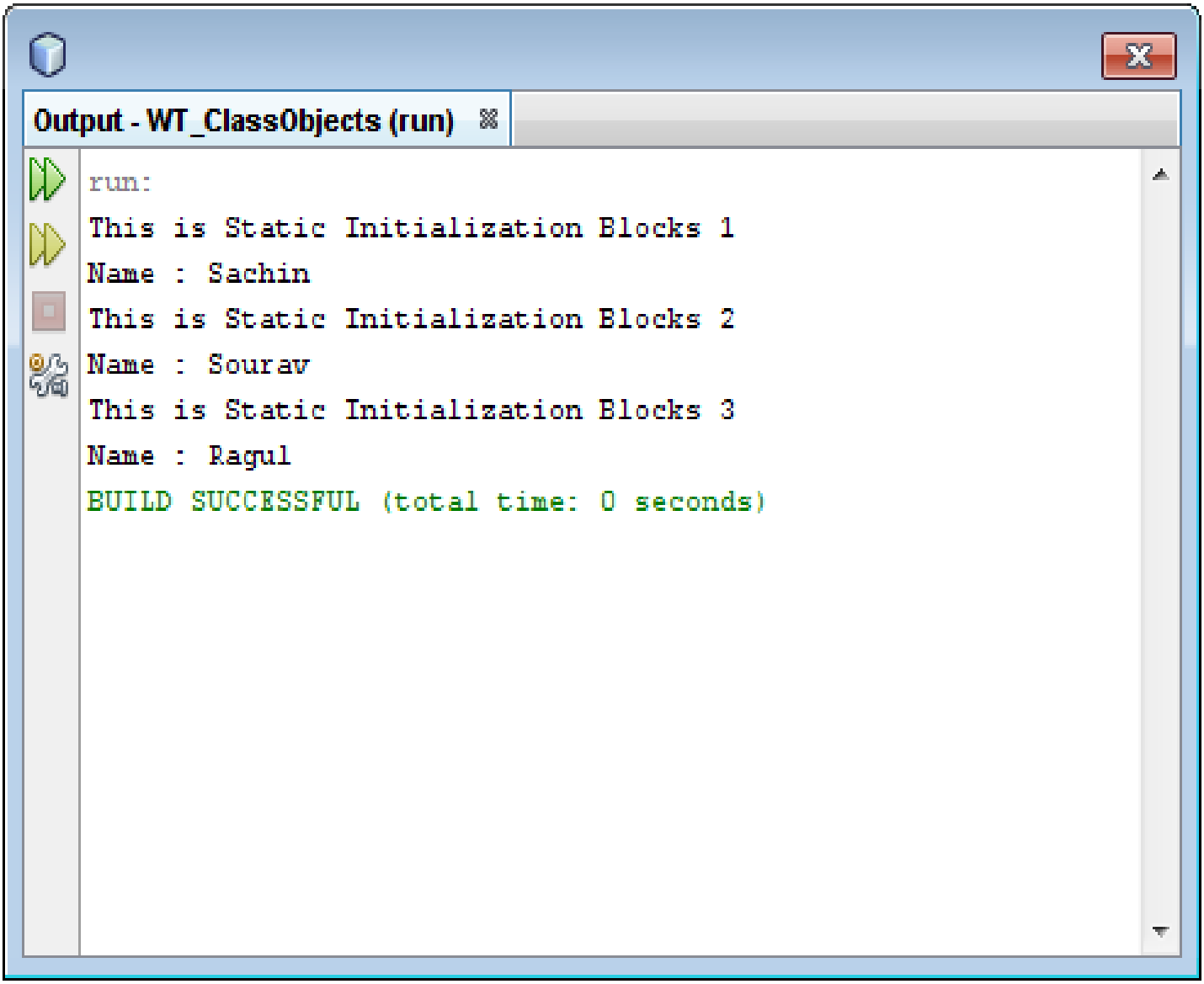
(StaticBlocks.java)

1. SOURCE CODE

```
public class StaticBlocks
{
// static variable declaration
static String name="Sachin";
// static Initialization blocks 1
    static
    {
        System.out.println("This is Static Initialization Blocks 1");
        System.out.println("Name : "+name);
        name="Sourav";
    }
// static Initialization blocks 2
    static
    {
        System.out.println("This is Static Initialization Blocks 2");
        System.out.println("Name : "+name);
        name="Ragul";
    }
// static initialization blocks 3
    static
    {
        System.out.println("This is Static Initialization Blocks 3");
```

```
        System.out.println("Name : "+name);
    }
// main method
    static public void main(String[] args)
    {
        // no need to create an object
    }
}
```

2. OUTPUT



VIII. STATIC BLOCKS-2

(StaticBlock.java)

1. SOURCE CODE

```
public class StaticBlock
{
static int s=500;
// 1st static block
    static
    {
        System.out.println("This is 1st static block...");
        System.out.println("s="+s);
        s+=500;
    }

// instance constructor
    public StaticBlock()
    {
        System.out.println("This is a default constructor...");
    }
// 2nd static block
    static
    {
        System.out.println("This is 2nd static block...");
        System.out.println("s="+StaticBlock.s);
        s+=4500;
    }
// static method
    static void disp()
    {
        System.out.println("This is static method...");
    }
```

```
// main method
    public static void main(String[] args)
    {
        StaticBlock obj=new StaticBlock();
    }
// 3rd static block
    static
    {
        System.out.println("This is 3rd static block...");
        System.out.println("s="+StaticBlock.s);
    // calling static method
        disp();
    }
}
```

2. OUTPUT