JAVA INTERFACE

EXISTING PROBLEM

- C++ supports multiple inheritances directly. But java does not support multiple inheritances directly.
- In java, a class can inherit only one super class at a time. It is not possible to inherit more than one super class (multiple super classes) at a time.

Example

Problem:

```
class one extends two, extends three
{
        // code
}
```

The above example is not allowed in java.

INTERFACE

- To overcome the above issues, java provides the new concept is called as interfaces. It gives the solution for multiple inheritances in both Java / C#.NET
- Interfaces are basically kind of a class. It contains only member's declaration & no implementation.
- It contains only abstract methods and final fields (constant variables) by default
- It is not possible to directly create an object for interface. With help of sub class object, we can call the interface methods.

POINTS

- Like class, interface is a block of code
- It has only method declaration and no method implementation. The method implementation must be given to the derived class under the public scope.
- Interfaces contain only abstract methods and final fields. Constructors and other methods are not allowed
- By default, all the variables are finals (constants) and all the methods are public (abstract methods).

CREATING AN INTERFACE

- Use the interface keyword to create an interface
- The access level for the entire interface is usually public.

Syntax

```
interface <name>
{
        // Constant Variable Declarations & Definitions
        // Method Declarations (No Implementation)
}
where,
interface ← is a keyword
name ← is a user defined name
```

Variable Declarations & Definitions:

Syntax

```
static final <type> <var-name>=initial value;
```

Example

```
static final int CODE=99;
static final float PI=3.14f;
static final String name="Ganesh";

                    (OR)
int CODE=99;
float PI=3.14f;
String name="Ganesh";
```

NOTE

- Note that, all the variables are implicitly final (constant variables)

Method Declarations

Syntax

```
return-type <method-name> (args);
```

Example

```
int result();
void disp(String name);
```

Note

- Note that, all the methods are implicitly abstract methods.

Example of Interface

```
interface Circle
{
        float PI=3.14f;         // constants
        void area();            // empty-method
}
```

Another Example

```
interface larea
{
        static final int PLAYERS=11;        // constants
        public abstract void sq_area();     // empty-method
        void result();                      // empty-method
}
```

Default Access Modifiers – (Interface Methods)

public, abstract for methods (by default)

Default Access Modifiers– (Interface Variables)

final, static for variables (by default)

DIFFERENCE BETWEEN CLASS AND INTERFACE

S.N	Class	Interface
1.	User defined type. It has member's declaration & definitions.	Block of code like class. It contains only method declaration. It has no method definition
2.	All the methods and variables are default by default.	All the methods are public by default. All the variables final by default.
3.	Creating an object is possible	Creating an object is not possible directly.
4.	It supports declaration and definition.	It supports declaration only. Method implementation must be given to derived class using public modifier.
5.	It supports constructors	It does not support constructors

DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

S.N	Interface	Abstract class
1.	It supports multiple inheritances	It does not support multiple inheritances
2.	All the methods in the interface are abstract methods by default (All the methods are empty. No code implementation)	It can contain abstract methods and non-abstract methods (instance / static methods). But it should contain atleast one abstract method.

3.	It has only constant variables	It includes local variables, instance variables, static variables and constant variables
4.	The interface keyword is used to create an interface	The abstract modifier is used to create abstract class and abstract methods
5.	The interface is inherited by using implements keyword	The abstract class is inherited by using extends keyword
6.	It does not provide the implementation of abstract class	It is possible to provide the implementation of interface. The interface methods must be defined using public modifier in abstract class Finally, sub class is required to call the methods of abstract class and interface
7.	It supports only public modifier	It supports public, private, protected modifiers, etc …

8.	**Example**
	interface Message
	{
	void info();
	}
		Example
		abstract class Test
		{
		abstract void info();
		void disp()
		{
		// user code
		}
		}

OBJECT CREATION

- Interface is an incomplete type. So creating an object for interface is not possible directly.
- But Indirectly we can get object through sub class object (object alias- assigning sub class object to interface object variable)
- So we can call the interface methods by using sub class (derived class) object or object alias of interface.
- Object alias for interface is possible.
  - Assigning a sub class object to interface object reference variable.

Syntax (Approach 1)

```
<interface-name> obj=<sub-class object>;
```

OR

Syntax (Approach 2)

```
< sub-class object>;
```

IMPLEMENTING INTERFACES

- Interfaces are used as “superclasses” whose properties are inherited by new classes (sub classes)
- Interfaces can be implemented by sub class using implements keyword.
- A sub class (derived class) can have any number of interfaces.

Implementing Interfaces

Syntax

```
sub-class <class-name> implements interface 1, interface2, … interface n
{
        // implementation of interface methods
        // implementation of sub class own methods
}
```

Extending Class & Implementing Interfaces

Syntax

```
sub-class <class-name> extends superclass implements interface1, interface2, … interface n
{
        // implementation of interface methods
        // implementation of sub class own methods
}
```

I. ACCESSING INTERFACE MEMBERS

(SimpleInterface.java)

1. SOURCE CODE

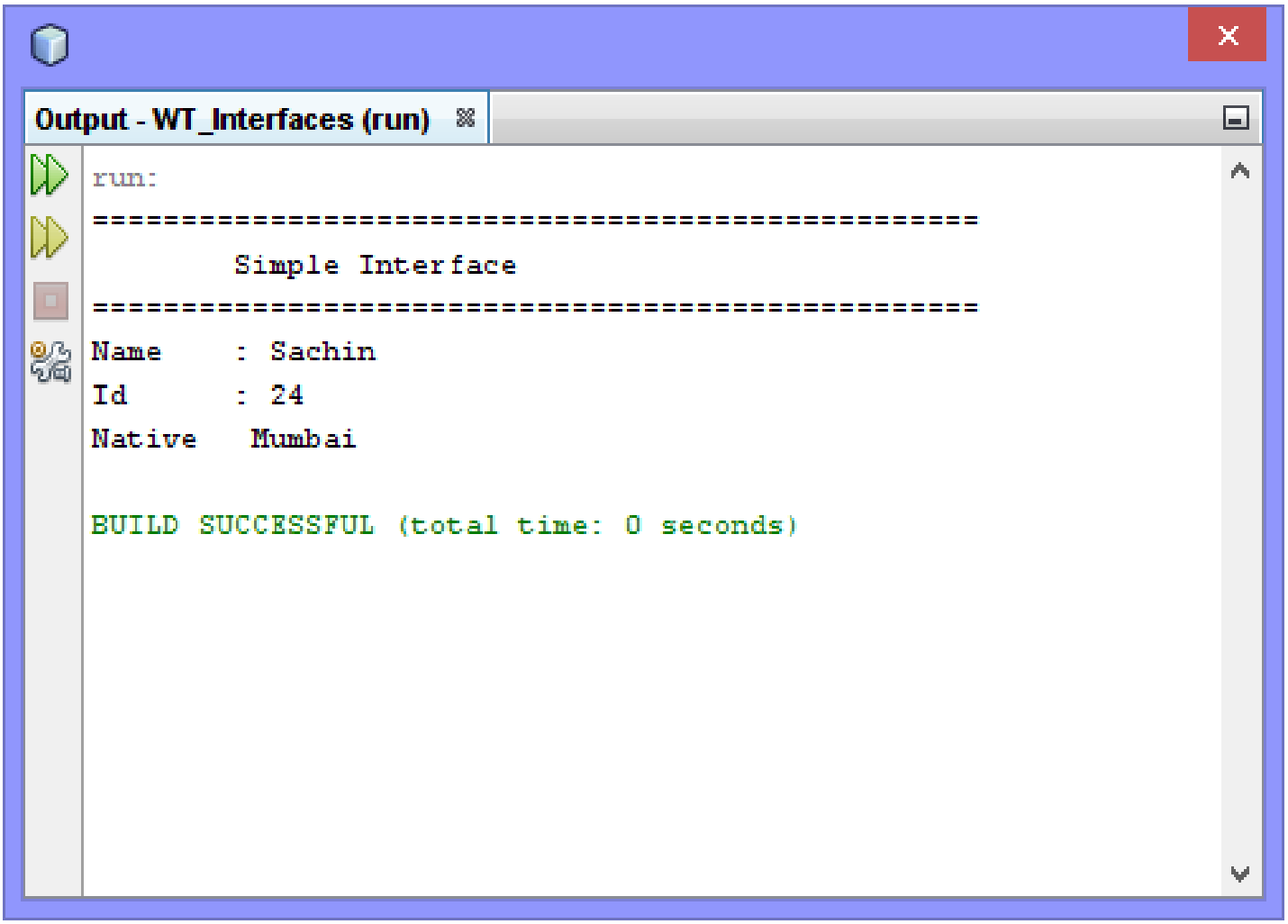
```
// define an interface 'Player'
interface Player
{
    String name="Sachin";        // constant by default
    int id=24;                   // constant by default
    void disp();                 // abstract method
}

public class SimpleInterface implements Player
{
// interface method
    public void disp()
    {
        System.out.println("Name \t: "+name);
        System.out.println("Id \t: "+id);
    }

// sub class Own method
    void info(String loc)
    {
        System.out.println("Native\t "+loc+"\n");
    }
// main method
    public static void main(String[] arr)
    {
        System.out.println("==================================");
        System.out.println("\tSimple Interface");
        System.out.println("==================================");
// creating sub class object
```

interface

sub class (derived class)

```
        SimpleInterface sobj=new SimpleInterface();
// accessing interface method by using sub class object
        sobj.disp();
// accessing sub class method using sub class object
        sobj.info("Mumbai");
    }
}
```

2. OUTPUT



II. IMPLEMENTING INTERFACES

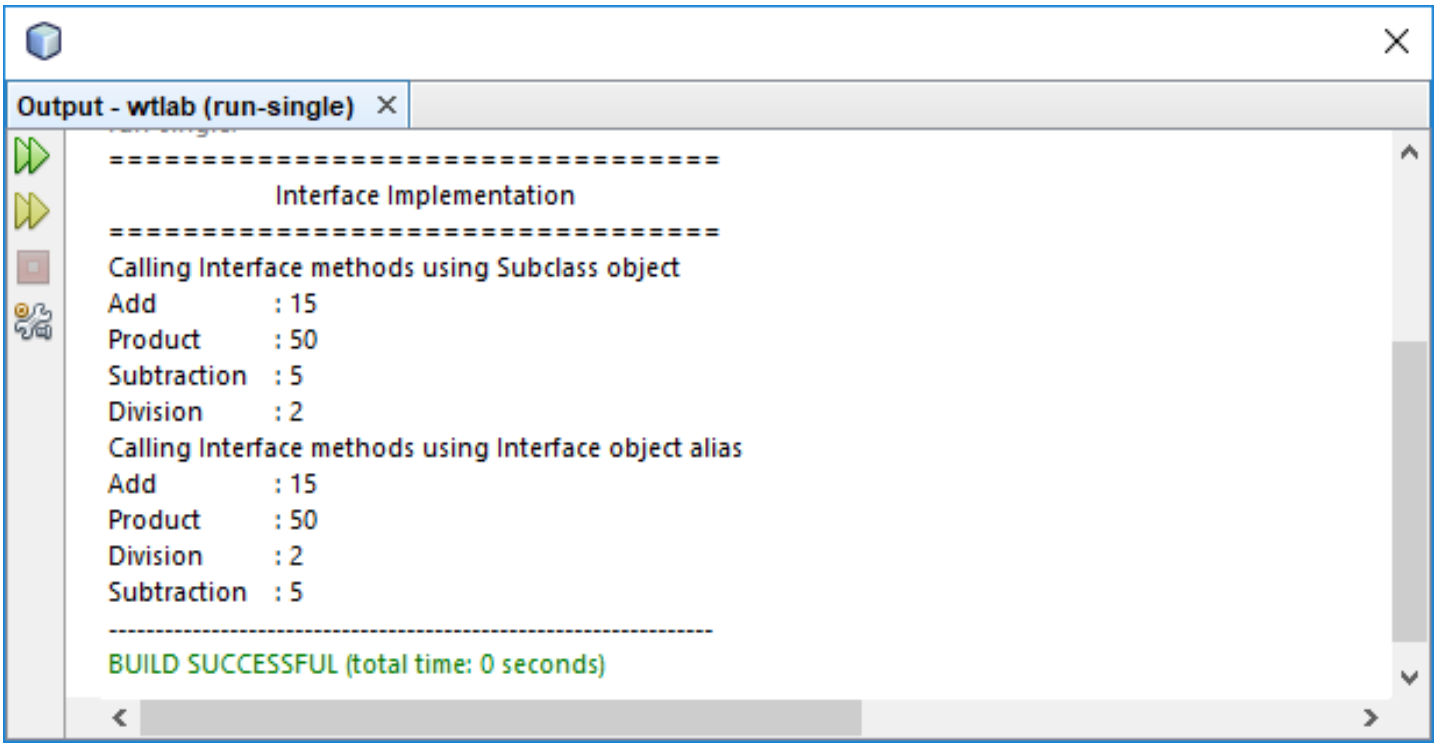
(Calc.java)

1. SOURCE CODE

```
// interface definition
interface Icalc
{
// abstract methods by default
      void add(int x,int y);
      void mul(int x,int y);
      void div(int x,int y);
      void sub(int x,int y);
}
// implementing an interface using implements keyword
public class JCalc implements Icalc
{
// implementation of add() of Icalc interface
   public void add(int a, int b)
   {
      System.out.println("Add \t\t: "+(a+b));
   }
// implementation of mul() of Icalc interface
   public void mul(int a, int b)
   {
      System.out.println("Product \t: "+(a*b));
   }
// implementation of div() of Icalc interface
   public void div(int a, int b)
   {
      System.out.println("Division \t: "+(a/b));
   }
```

```
// implementation of sub() of Icalc interface
    public void sub(int a, int b)
    {
        System.out.println("Subtraction \t: "+(a-b));
    }
// main method
    public static void main(String[] args)
    {
        System.out.println("==============================");
        System.out.println("\tInterface Implementation");
        System.out.println("==============================");
// creating a sub class object
        JCalc obj=new JCalc();
// assigning subclass object to interface object variable
        Icalc cc=obj;
// calling interface methods using sub class object
        System.out.println("Calling Interface methods using Subclass object");
        obj.add(10,5);
        obj.mul(10,5);
        obj.sub(10,5);
        obj.div(10,5);
// calling interface methods using interface object
        System.out.println("Calling Interface methods using Interface object
alias");
        cc.add(10, 5);
        cc.mul(10,5);
        cc.div(10, 5);
        cc.sub(10, 5);
        System.out.println("----------------------------------------------");
     }
}
```

2. OUTPUT



III. MULTIPLE INHERIATNCE USING INTERFACES

(MultipleInheritance.java)

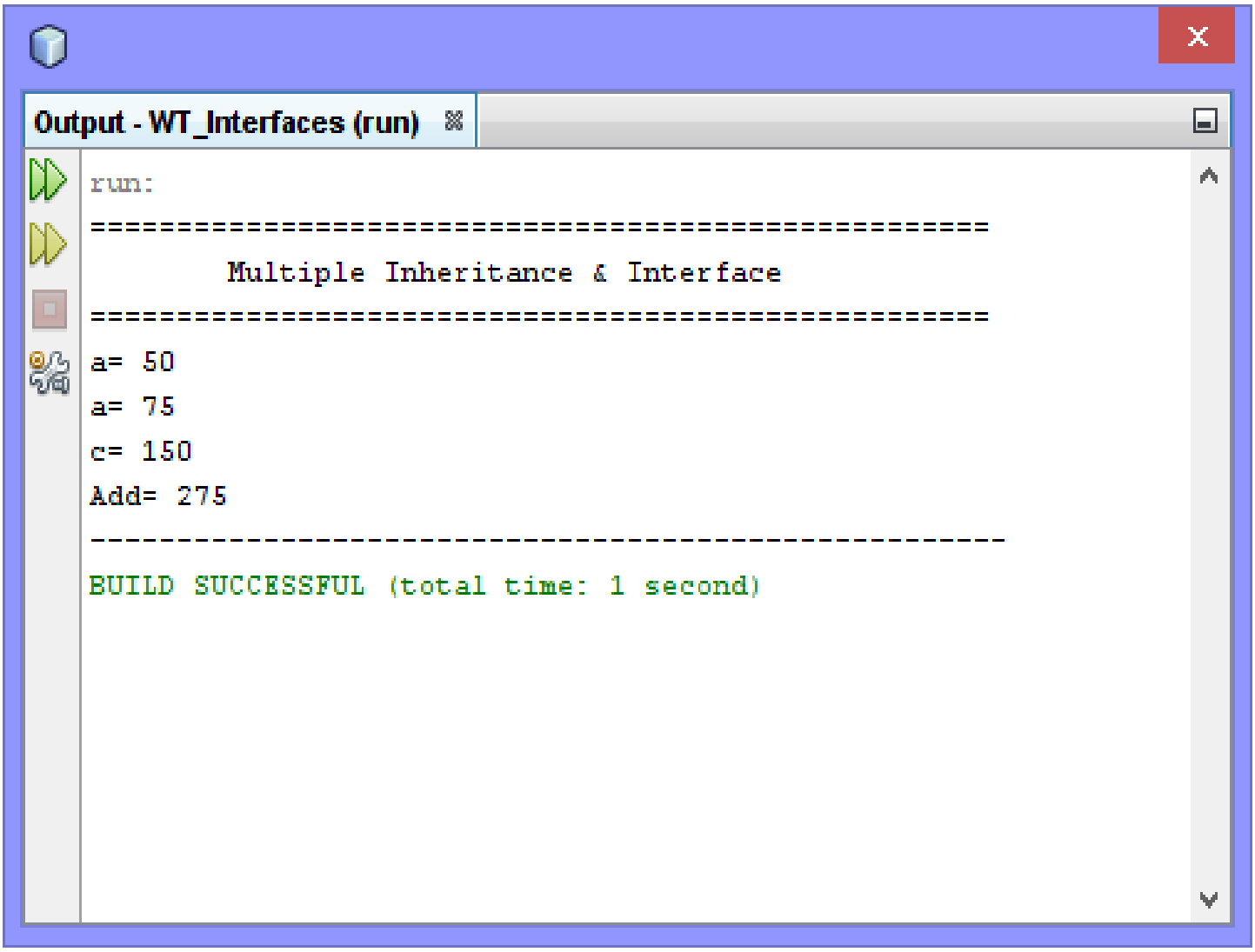
1. SOURCE CODE

```
// super interface A
interface A
{
    int a=50;                   // constant
    void printA();              // empty-method
}


// super interface B
interface B
{
    int b=75;                   // constant
    void printB();              // empty method
```

```
}
// super class C
class C
{
    int c=150;
    void printC()
    {
        System.out.println("c= "+c);
    }
}
// inheriting one super class & two super interfaces
public class MultipleInheritance extends C implements A,B
{
 // implementation of interface method 1
    public void printA()
    {
        System.out.println("a= "+a);
    }
// implementation of interface method 2
    public void printB()
    {
        System.out.println("a= "+b);
    }
// implementation of sub class method
    void sum()
    {
        int r=a+b+c;
        System.out.println("Add= "+r);
    }
// main method
    public static void main(String[] arr)
    {
        System.out.println("================================");
```

```
        System.out.println("\tMultiple Inheritance & Interface");
        System.out.println("==============================");
        MultipleInheritance obj=new MultipleInheritance();
// calling methods of super interface & super class using sub class object
        obj.printA();
        obj.printB();
        obj.printC();
        obj.sum();
        System.out.println("-----------------------------------------------------");
    }
}
```

2. OUTPUT



EXTENDING INTERFACES

- Like class, one interface can extend more than one interface using extends keyword
- An interface can be subinterfaced from other interfaces. The new interface (sub interface) will inherit all the members of the super interfaces
- This is done by using extends keyword.

Note

- While interfaces are allowed to extend to other interfaces, subinterfaces cannot define the methods declared in the super interfaces. After all, sub-interfaces are still interfaces not classes.
- It is important to note that, when an interface extends two or more interfaces, they are separated by comma operator (,).
- Interface cannot extend classes.
- Interfaces can have only abstract methods and final fields (constant variables)

SYNTAX

```
interface sub extends super 1, super 2, … super n
{
        // body of the sub-interface
}
```

Note

- Here sub-interface just extended the super interfaces.
- Only sub class (derived class) have to provide the necessary implementation of all methods of super interfaces and sub interfaces.

IV. EXTENDING INTERFACES

(Exdinterefaces.java)

1. SOURCE CODE

```
// super interface 1
interface Person
{
   String name="Krishna";          // constant variable 1
   int id=33;                      // constant variable 2
}
// super interface 2
interface Employee
{
   void info();                    // empty-method()
}
// sub interface
interface sub extends Person, Employee
{
   void add_info(double s, String l); // empty-method()
}
// sub class inherits the properties of super & sub interfaces
public class Exdinterefaces implements sub
{
// implementation of super interface method
   public void info()
   {
      System.out.println("Name\t\t: "+name);
      System.out.println("Id  \t\t: "+id);
   }
// implementation of sub interface method
   public void add_info(double sal, String loc)
   {
      System.out.println("Salary      \t: "+sal);
```

```
        System.out.println("Location    \t: "+loc);
    }
// main method
    public static void main(String[] args)
    {
        System.out.println("==================================");
        System.out.println("\t\tExtending Interfaces");
        System.out.println("==================================");
        Exdinterefaces obj=new Exdinterefaces();
// call all the methods of super and sub interface methods using sub-object
        obj.info();
        obj.add_info(33000, "Kanchipuram");
        System.out.println("-----------------------------------------------------");

    }
}
```

2. OUTPUT