JAVA INHERITANCE

INHERITANCE

- Inheritance is pure object oriented programming technique.
- Inheritance is a process of creating a new class from old class.
- Constructing / Deriving a class from an existing class.
- New class : Child class / Sub class / Derived class
- Old class : Parent class / Super class / Base class

Implementation

- In java, inheritance is implemented in two ways. They are:
  1. Inheriting classes (Extending classes)
  2. Implementing interfaces (Implementing interfaces)

Characteristics of Inheritance

1. A class can't inherit more than one super class at a time.
2. A class can inherit more than one interface at a time.
3. Sub class can access only the non-private members of the super class.
4. All the methods declared within super interface must be defined with public modifier in sub class.

Super Class : Most general class

Sub Class : Most dependent class

Construction of new class (sub class)

- Java provides two options for creating a new class (sub class) from an existing class. They are:
    1. Deriving a sub class from super class
    2. Deriving a sub class from super interface

Syntax of Sub class

1. Deriving a sub class from super class

```
class <sub-class-name> extends <super-class-name>
{
        // variable decl.
        // method decl.
}
```

Example

```
class sample
{
        ...
}
class test extends sample
{
        void disp()
        {
                …
        }
}
```

extends

- This reserved keyword is used to inherit the copy of super class's properties. The properties can be non-private members of super class (Ex. variables, methods, etc, ...)
- It is an important to note that, it is used to permit only one super class at a time.

2. Deriving a sub class from super interface

Syntax

```
class <sub-class-name> implements <interface-name>
{
        // variable decl.
        // method decl.
}
```

Example

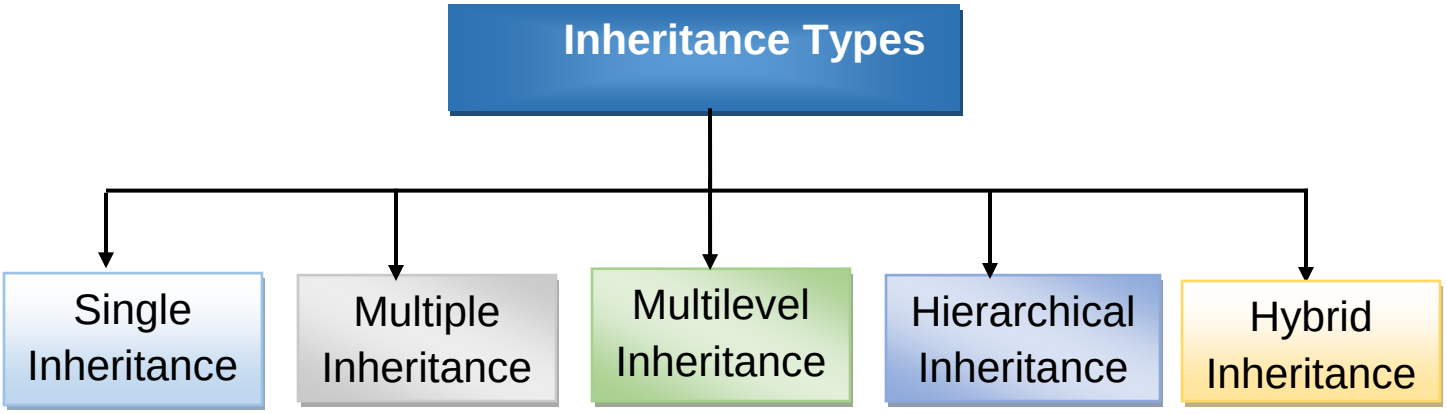
```
interface Icalc                 // super interface
{
        void add();             // method declaration
}
class test extends Icalc
{
        public void add()       // method definition
        {
                …
        }
}
```

implements

- This reserved keyword is used to inherit the copy of super interface 's properties.
- It is used to allow 'n' number of interfaces.

TYPES OF INHERITANCE

- In java, inheritance is classified as five types. They are
    1. Single Inheritance
    2. Multiple Inheritance
    3. Multilevel Inheritance
    4. Hierarchical Inheritance
    5. Hybrid Inheritance



Single Inheritance : Only one super class & sub class.

Multiple Inheritance : Several Super classes.

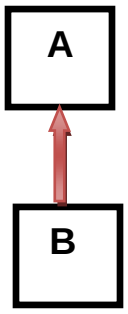
Multilevel Inheritance : Derived from a sub class

Hierarchical Inheritance : Only one super class & Several Sub classes

Hybrid Inheritance : involving at least one form of inheritance.

1. SINGLE INHERITANCE

- Creating a new class (sub class) from only one super class.
- Like 1-1 mapping.



A← Super Class

B← Sub Class.

Fig: Single Inheritance

1. EXAMPLE OF SINGLE INHERITANCE

(B.java)

1. SOURCE CODE

```
// super class definition
class A
{
        int x=25;
        void print_a()
        {
                System.out.println("x = "+x);
        }
}
// sub class definition
class B extends A          // Inherit the properties of super class A
{
    int y=50;
```

```
// own method of sub class
    void print_b()
    {
// calling super class method
        print_a();
        System.out.println("y = "+y);
    }
// own method of sub class
    void sum()
    {
        int s=x+y;
        System.out.println("Sum = "+s);
    }
}
// separate main class
public class Single {
    public static void main(String args[])
    {
        System.out.println("============================");
        System.out.println("\t\tSingle Inheritance");
        System.out.println("============================");
// sub class (derived) class object creation
        B obj=new B();
// calling super and sub class instance methods using object
        obj.print_b();
        obj.sum();
    }
}
```

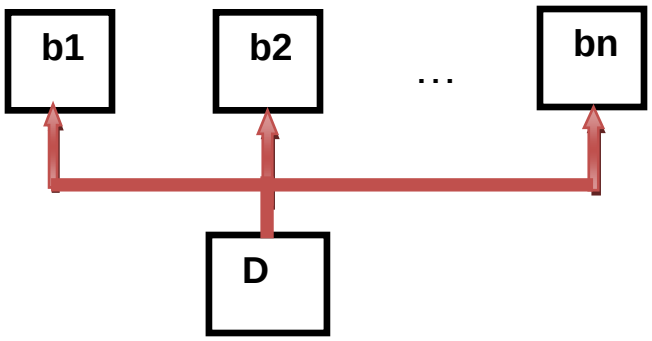
2. OUTPUT

Output - WT_Class_Object (run)

```
run:
=====================================================
                  Single Inheritance
=====================================================
x = 25
y = 50
Sum = 75
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. MULTIPLE INHERITANCE

- The process of creating a new class from more than one super class (several super classes) is called as multiple inheritance.
- just like as 1-many.



Where,

b1,b2..bn ← Super Classes D ← Sub Class

Fig: Multiple Inheritance

Notable point

- C++ gives direct support for multiple inheritances. But in java, it is not possible to inherit more than one super class at a time
- Java does not directly support multiple inheritances. But it gives indirect support of multiple inheritances. This is implemented with help of interfaces in java.

Interface

- It is a new concept of java. It gives the solution for multiple inheritances in both Java / C#.NET
- An interface is basically a kind of class. But it contains abstract methods & final fields by default
- It contains only method declaration. We must provide the method implementation in a sub class.

- It is not possible to create an object for interface in java. With help of sub class object, we can call the interface methods.
- By default, all the methods declared inside an interface are abstract methods even the abstract keyword is not used and by default.
- By default, all the methods in an interface are public.
- By default, all the variables defined inside an interface are final variables (constant variables) even final keyword is left.
- Implements keyword is used to inherit the properties of interface.

DIFFERENCE BETWEEN CLASS AND INTERFACE

S.N	Class	Interface
1.	User defined type. It has member's declaration & definitions.	Block of code like classes. It contains only method declaration. It has no method definition
2.	All the methods & variables are friendly 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

Syntax of Interface

```
interface <interface-name>
{
        // constant variable definition
        // method declarations only (no definition)
}
```

Example of Interface

```
interface Circle
{
        float PI=3.14f;
        void area();
}
```

Interface Member Access

- Java provides two options for calling the methods of interface
    1. Using interface object through object alias (assigning sub class object to interface object)

       OR

    2. Using sub class object.

Interface Object Creation

- Creating an object for interface is not possible directly
- But indirectly we can create an object by just assigning the object of sub class to interface object variable.

Syntax

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

Example

```
interface Calc
{
	void add();
}
public class SubImp implements Calc
{
	public void add()
	{
	}
	public static void main(String[] args)
	{
// object creation for sub class
		SubImp obj=new SubImp();
// interface object creation
		Calc cc=obj;
// call interface methods using interface object (not sub class object)
		cc.add();
	}
}
```

2. EXAMPLE OF MULTIPLE INHERITANCE

(Multiple.java)

1. SOURCE CODE

```
// super interface 1
interface Person
{
   String name="Sachin";            // constant variables
   final int age=25;
   void person_info();              // method declaration
}
// super interface 2
interface Employee
{
   int id=233;
   String cmp="Google";
   void emp_info();
}
// sub class
class D implements Person, Employee
{
   double income=95000.45638;
// super interface 1 method
   public void person_info()
   {
      System.out.println("Name \t: "+name);
      System.out.println("Age \t: "+age);
   }
// super interface 2 method
   public void emp_info()
   {
      System.out.println("Id \t: "+id);
      System.out.println("Company\t: "+cmp);
```

Inherits two super interfaces Person, Employee

```
    }
// own method of sub class
    void disp()
    {
        person_info();
        emp_info();
        System.out.printf("Income\t: %8.2f\n",income);
    }
}
// separate main class
public class Multiple {
    public static void main(String a[])
    {
         System.out.println("================================");
         System.out.println("\tMultiple Inheritance");
         System.out.println("================================");
// object creation for sub class
         D obj=new D();
         obj.disp();
    }
}
```

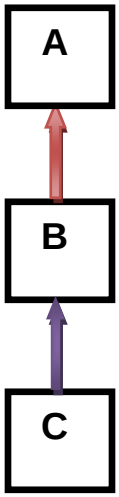
2. OUTPUT

Output - WT_Class_Object (run)

```
run:
==============================================
        Multiple Inheritance
==============================================
Name    : Sachin
Age     : 25
Id      : 233
Company : Google
Income  : 95000.46
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. MULTILEVEL INHERITANCE

- The process of creating a new class from another derived class or sub class (already inherited sub-class) is called as multilevel inheritance.
- **Intermediate class**
  - A same class acts as super class on one side & sub class on another side.



Where, A← Super class

B←Sub class (Intermediate class)

C←Sub class

Fig: Multilevel Inheritance

3. EXAMPLE OF MULTILEVEL INHERITANCE

(Multilevel.java)

1. SOURCE CODE

```
// super class definition
class P
{
   int a=25;                    // default modifier
   void printP()
   {
      System.out.println("A = "+a);
   }
}
// sub class (Intermediate class): inherit one copy of class P
class Q extends P
{
   int b=25;                    // default modifier


   void printQ()
   {
      System.out.println("B = "+b);
   }
}
// sub class: inherit one copy of Q class using extends keyword
class R extends Q
{
   int c=50;                    // default modifier
   void printR()
   {
      System.out.println("C = "+c);
   }
   void sum()
```

```
    {
        int res=a+b+c;
        System.out.println("Sum = "+res);
    }
}
// separate Main class
public class Multilevel {
    public static void main(String a[])
    {
        System.out.println("==============================");
        System.out.println("\tMultilevel Inheritance");
        System.out.println("============================");
// object creation for sub class
        R obj=new R();
// call own method & super class methods using sub class object
        obj.printP();
        obj.printQ();
        obj.printR();
        obj.sum();
        System.out.println("-----------------------------------------");
    }
}
```

2. OUTPUT

Output - WT_Class_Object (run)

```
run:
==============================================
        Multilevel Inheritance
==============================================
A = 25
B = 25
C = 50
Sum = 100
----------------------------------------------
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. HIERARCHICAL INHERITANCE

- The process of creating several new classes from only one super class / base class is called as hierarchical inheritance.

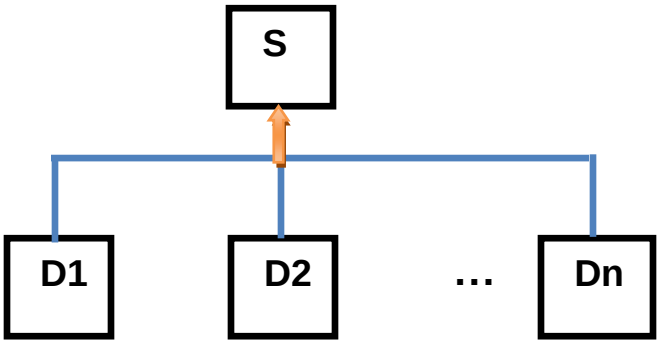


Fig: Hierarchical Inheritance

4. EXAMPLE OF HIERARCHICAL INHERITANCE

(Hierarchical.java)

1. SOURCE CODE

```
// super class definition
class S
{
    int a=35;
    void printS()
    {
        System.out.println("a ="+a);
    }
}
// subclass 1: inherit the copy of S(super class)
class Sub1 extends S
{
    int d1=50;
```

```
    void printS1()
    {
        printS();
        System.out.println("d1 ="+d1);
    }
}
// subclass 2: inherit the copy of S (super class)
class Sub2 extends S
{
    int d2=75;

void printS2()
    {
        printS();
        System.out.println("d2 ="+d2);
    }
}
// subclass 3: inherit the copy of S (super class)
class Sub3 extends S
{
    int d3=150;
    void printS3()
    {
        printS();
        System.out.println("d3 ="+d3);
    }
}
// separate main class
public class Hierarchical
{
    static public void main(String[] args)
    {
         System.out.println("==========================");
```

```
        System.out.println("\tHierarchical Inheritance");
        System.out.println("==========================");
// sub class 1 object creation
        Sub1 obj1=new Sub1();
        System.out.println("--------------Sub Class 1------------------");
        obj1.printS1();
// sub class 2 object creation
        Sub2 obj2=new Sub2();
        System.out.println("--------------Sub Class 2------------------");
        obj2.printS2();
// sub class 3 object creation
        Sub3 obj3=new Sub3();
        System.out.println("--------------Sub Class 3------------------");
        obj3.printS3();
    }
}
```

2. OUTPUT

Output - WT_Class_Object (run)

```
run:
==============================================
        Hierarchical Inheritance
==============================================
--------------Sub Class 1-------------------
a =35
d1 =50
--------------Sub Class 2-------------------
a =35
d2 =75
--------------Sub Class 3-------------------
a =35
d3 =150
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. HYBRID INHERITANCE

- The process of creating a new class involving at least one form of inheritance (single, multiple, multilevel & hierarchical) and one or more number of super classes / base classes are called as hybrid inheritance.

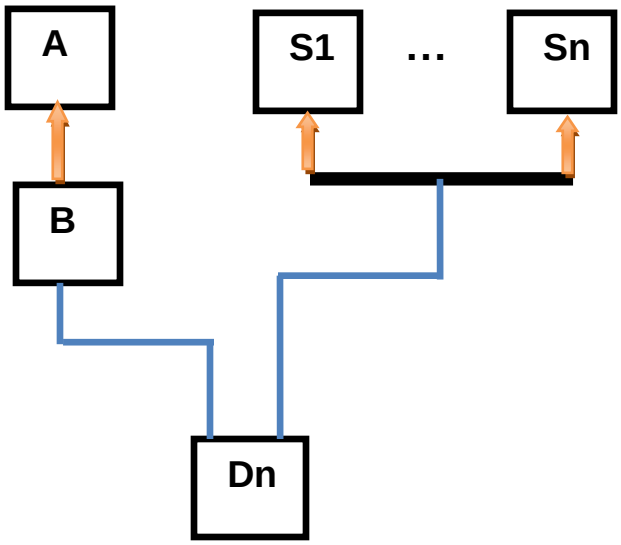


Figure. Hybrid Inheritance

5. EXAMPLE OF HYBRID INHERITANCE

(Hybrid.java)

1. SOURCE CODE

```
// super class definition
class X
{
    int a=191;
    void printX()
    {
        System.out.println("a = "+a);
    }
```

```
}
// single Inheritance
class SI extends X
{
    int b=283;
    void print_SI()
    {
        printX();                    // call the base class method
        System.out.println("b = "+b);
    }
}
// super interface (super class)
interface SS
{
    int c=100;
    void print_SS();
}
// new sub class(hybrid class)
class SBI extends SI implements SS
{
    int d=308;
    public void print_SS()
    {
        System.out.println("c = "+c);
    }
    // own method 1 of sub class
    void print_SBI()
    {
        print_SS();                      // call the interface method
        System.out.println("d = "+d);
    }
    // own method 2 of sub class
    void sum()
```

```
    {
        int res=a+b+c+d;
        System.out.println("Sum = "+res);
    }
}
// separate main class
public class Hybrid
{
    public static void main(String[] args)
    {
        System.out.println("============================");
        System.out.println("\tHybrid Inheritance");
        System.out.println("============================");
// object creation for sub class
        SBI obj=new SBI();
// calling instance methods using object
        obj.print_SI();
        obj.print_SBI();
        obj.sum();
    }
}
```

2. OUTPUT

Output - WT_Class_Object (run)

```
run:
==============================================
        Hybrid Inheritance
==============================================
a = 191
b = 283
c = 100
d = 308
Sum = 882
BUILD SUCCESSFUL (total time: 0 seconds)
```

FINAL VARIABLES AND METHODS

Final Modifier

- This modifier is used to create constant variable, constant method and constant class in java.

1. Final Variable (Constant Variable)

- In java, constants are defined with help of final modifier. Final is a new modifier in java.
- An instance variable declared with the keyword final is called as final variables. The final variables are equivalent to const qualifier in c++.
- The value of the final variable will not be changed during the program execution.
- It is used to assign the constant value to a variable
- It is conventional to write the final variable in uppercase.

Syntax

```
final <type> <var.name>=initial value;
```

Example

```
final int MARKS=100;
final float PT=3.14
```

2. Final Method (Constant Method)

- If an instance method is defined with final modifier, then it is called as final method / constant method.
- Final methods are not inherited in sub class (derived class).
- Final methods can't be overridden by subclasses.

Example

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

3. Final Class (Constant Class)

- If a class is defined with final modifier, then it is called as final class / constant class.
- Final class can't be extended in sub classes. Because it is a constant class.

Example

```
final class M
{
        // code
}
```

6. EXAMPLE OF FINAL CLASS

(sony.java)

1. SOURCE CODE

```
final class Ericsson
{
	// user code
}
class sony extends Ericsson
{
	void disp()
	{
	System.out.println("Never Executed...");
	}
}
```

2. OUTPUT

Sub class can't have final class.

SUMMARY

Final Variable	**:**	Its value can't be changed
Final Method	**:**	It can't be overridden further in sub class
Final Class	**:**	It can't be extended further in sub class.

ABSTRACT CLASS

- If a class is defined with abstract modifier & has abstract method with no implementation, then it is called as abstract class
- It contains only abstract method declarations. Method implementations must be provided in sub class.
- An abstract class is a class that can't be instantiated (Creating a direct object is not possible)
- Abstract class contains at least one abstract method & one or more normal methods
- Abstract modifier is used to define both abstract class and abstract methods
- Abstract constructors, abstract variables and abstract static methods can't be created.
- Abstract methods must be called by using the sub class (derived class) object

POINTS

- Like interface, abstract class contains only declarations of abstract methods not implementation
- It is important to note that, Abstract methods must be implemented in sub class

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	It can contain abstract methods and non-abstract methods (instance / static methods)
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.	Here methods are public by default	It supports public, private, protected modifiers, etc …

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

Member Access

- Java provides two options for calling the methods of abstract class

  3. Using abstract class object through object alias (assigning sub class object to abstract class object)

OR

  4. Using sub class object.

Object Creation

- Creating an object for abstract class is not possible directly.
- But indirectly we can create an object by just assigning the object of sub class to interface object variable.

Syntax

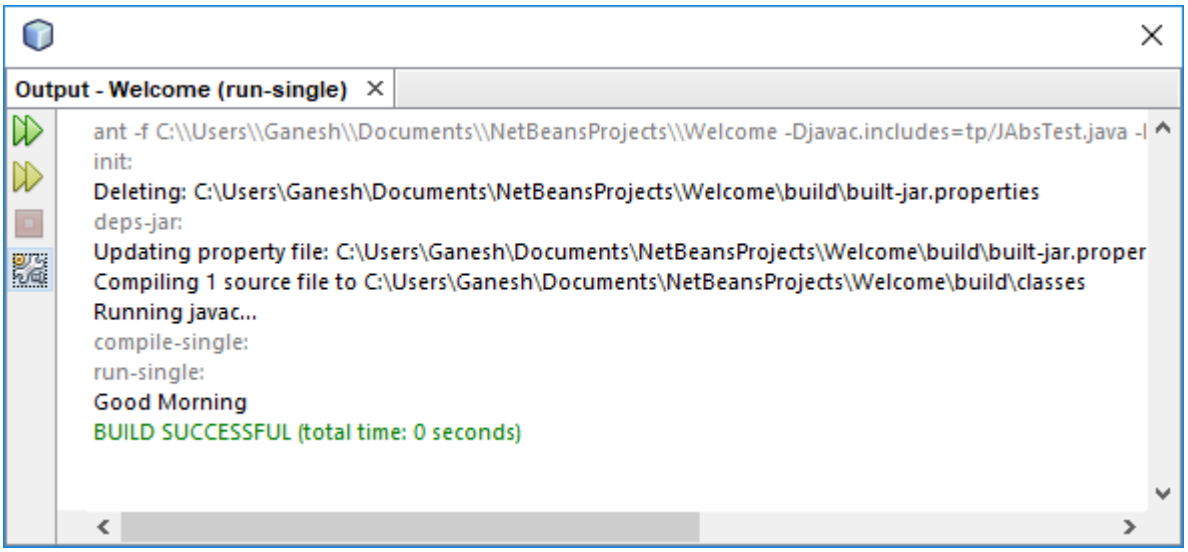
```
<abstract class-name> obj=<sub-class object>
```

Example

Source Code

```
// abstract class definition
abstract class Welcome
{
        abstract void disp();
}
// sub class definition
public class JAbsTest extends Welcome
{
    void disp()
    {
            System.out.println("Good Morning");
    }
    public static void main(String[] args)
    {
// object creation for sub class
        JAbsTest obj=new JAbsTest();
// abstract class object creation
        Welcome wc=obj;
// call interface methods using interface object (not sub class object)
        wc.disp();
    }
}
```

Output



6. EXAMPLE OF ABSTRACT CLASS

(Demo.java)

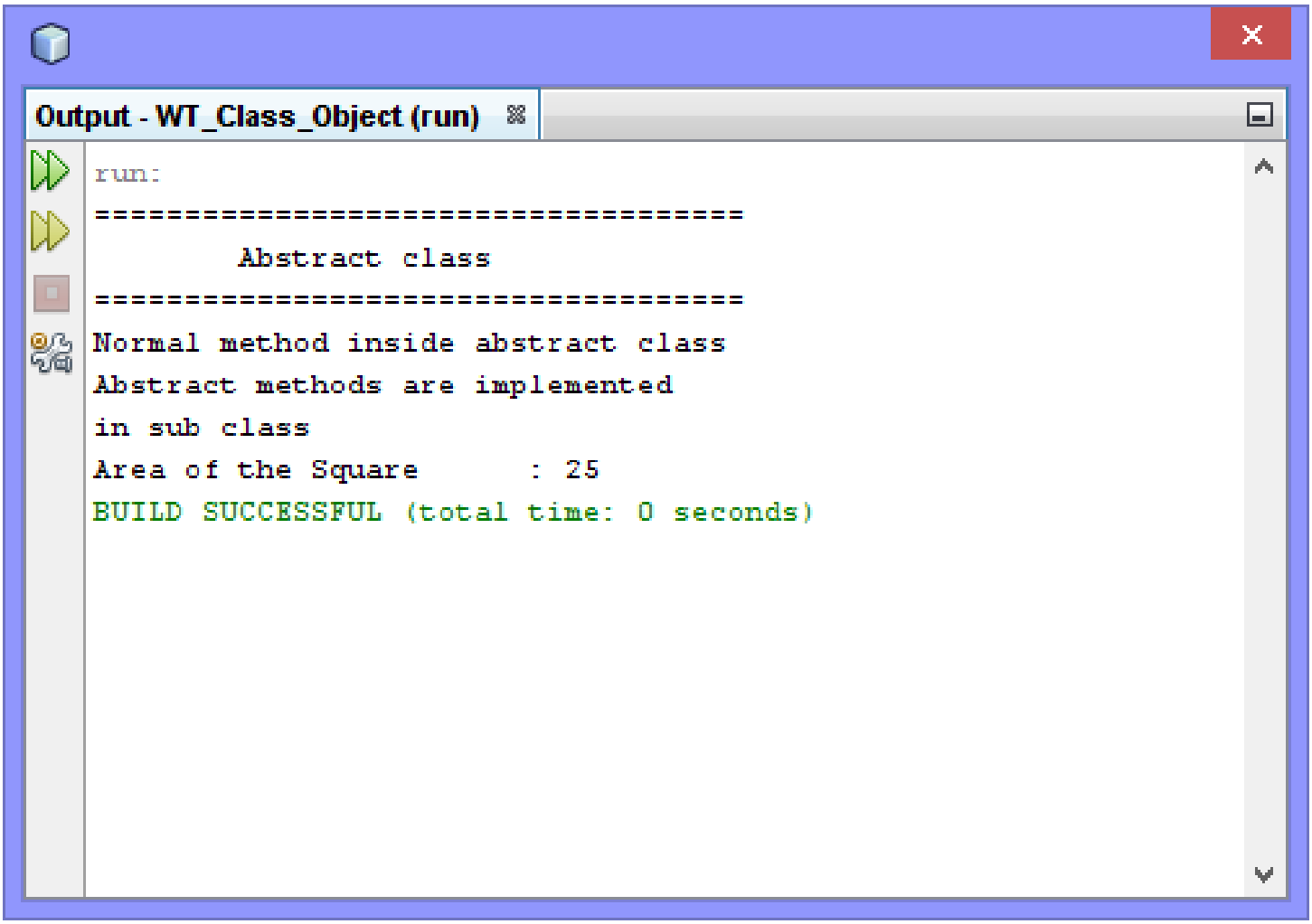
1. SOURCE CODE

```
// abstract class
abstract class first
{
// instance method: abstract class allows normal instance method
    void disp()
    {
        System.out.println("Normal method inside abstract class");
    }

// abstract method
    abstract void area(int x);
}
```

```
// sub class
class D extends first
{
// implementation of abstract method in derived class
    void area(int x)
    {
        int s=x*x;
        System.out.println("Abstract methods are implemented\nin sub class");
        System.out.println("Area of the Square\t: "+s);
    }
}
// main class
public class Demo
{
    public static void main(String[] args)
    {
        System.out.println("=====================================");
        System.out.println("\tAbstract class");
        System.out.println("=====================================");
// object creation for sub class
        D obj=new D();
// calling abstract class methods using sub class object
        obj.disp();
        obj.area(5);
    }
}
```

2. OUTPUT



POINTS TO REMEMBER ABOUT ATSTRACT CLASS

- Abstract class contains abstract methods as well as normal methods
- Creating an object for abstract class is not possible directly
- If any class (sub class) extends an abstract class, then it must be implement all the abstract methods of that abstract class.

BENEFITS OF INHERITANCE

- Enhancement of base class
- It saves the storage space and time
- It reduces the code length
- It allows the reusability of code
- It increases the reliability of code.