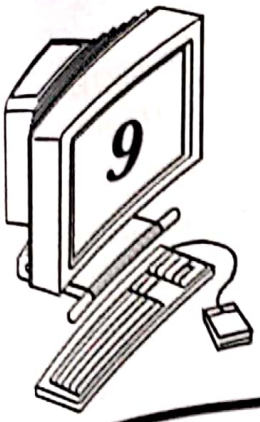


Interfaces

9.1 Introduction

Java does not support **multiple inheritance**. But, a large number of real-life applications require the use of multiple inheritance. Therefore, java provides an alternate approach known as **interfaces** to support the concept of multiple inheritance.

The interface in java is also a mechanism to achieve **fully abstraction**. Interface is a collection of abstract data members such as methods. All the members of an interface are implicitly **public** and **abstract**. The method defined in an interface do not have their implementation and only specify the parameters they will take and the type of values they will return. An interface is always **implemented in a class**. The class implementing an interface must define all the methods contained inside an interface, otherwise Java compiler given an error message. Interface in Java is equivalent to an abstract base class. You cannot instantitate an object through an interface, but you can offer a set of functionalities that is common to several different classes.

9.2 What is an Interface?

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default **abstract** (only method signature, no body). Also, the variables declared in an interface are **public**, **static** & **final** by default.

Interfaces are used for **abstraction**. Since methods in interfaces does not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not support multiple inheritance, using interface we can achieve this as a class can implement more than one interfaces.

Note :

The java compiler adds automatically public and abstract keywords before the interface method and public, static and final keywords before data members.

9.3 Reasons to use Interface

There are three reasons to use interface.

1. It is used to achieve fully abstraction.
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling.

9.4 Defining an Interface

Interfaces are syntactically similar to classes.

Syntax is :

```
Interface interfacename
{
    variables declaration;
    methods declaration;
}
```

Where

interface is keyword

interfacename is any valid java identifier.

Methods are declared using only their return type and signature. They are, abstract methods. We know now that in an interface, no method can have an implementation. All variables declared in the interface definition are **constant**, therefore, they should be initialized with a **constant value.**

Example 9.1

```
Interface student
{
    int num = 40;
    void display( );
}
```

In this example, student is the name of interface. It contains one constant **num** having value equal to 40 and one method **display( )**. Interface variables are public, static and final by default and methods are public and abstract as shown in figure 9.1.

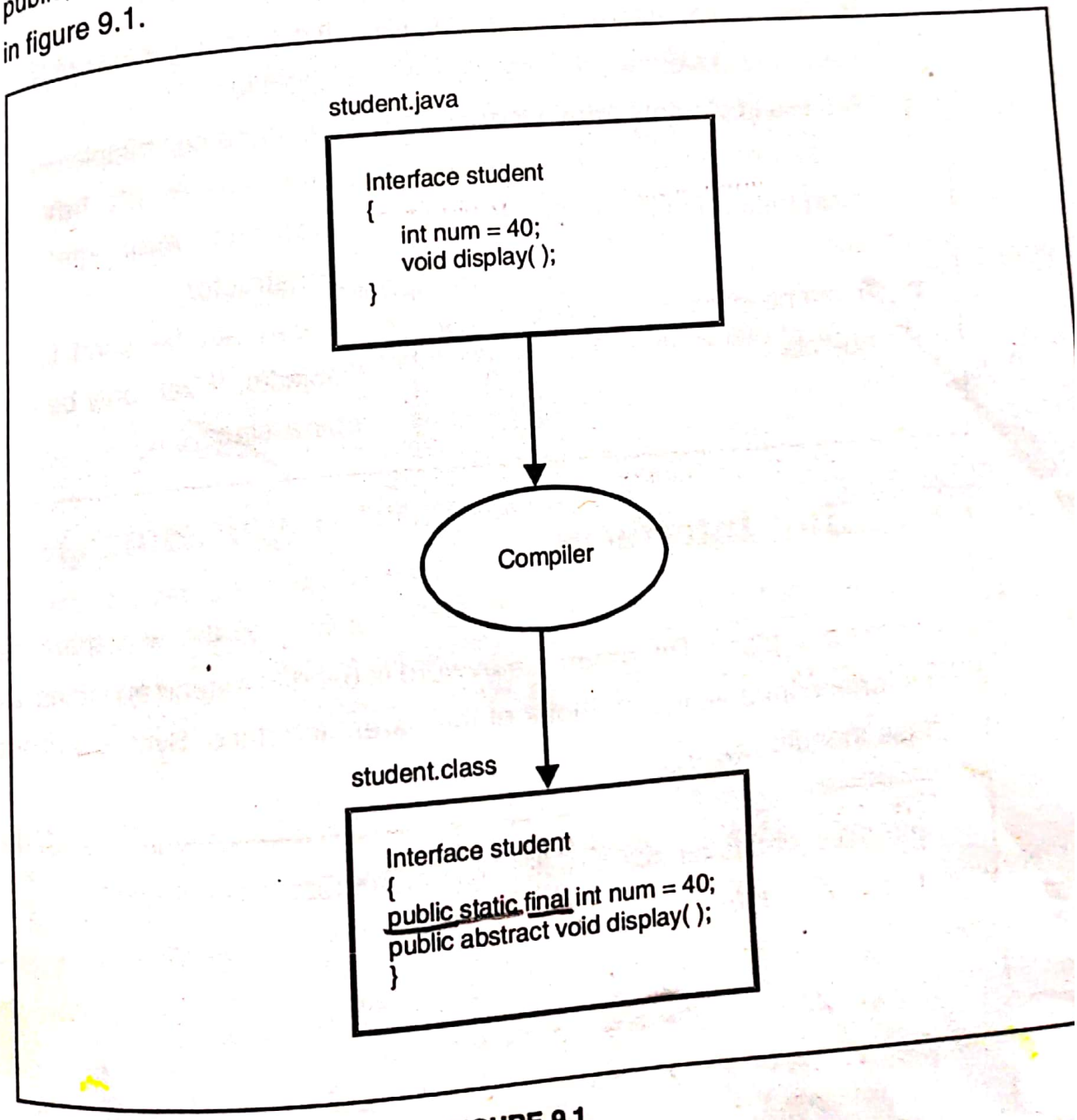


FIGURE 9.1

9.5 Differences between Class and Interface

Differences between class and interface are shown in table 9.1.

TABLE 9.1

Class	Interface
1. Class can have abstract and non-abstract methods.	1. Interface can have only abstract methods.
2. Class doesn't support multiple inheritace.	2. Interface supports multiple inheritance.
3. Class can have final, non-final, static and non-static variables.	3. Interface has only static and final variables.
4. It can use various access specifiers like public, or protected.	4. It can only use the public access specifier.
5. A class implements the interface.	5. Interface can't implement class.
6. Class can have static methods, main method and constructor.	6. Interface can't have static methods, main method and constructor.
7. It can be instantiated by declaring objects.	7. It can not be used to declare objects. It can only be inherited by a class.

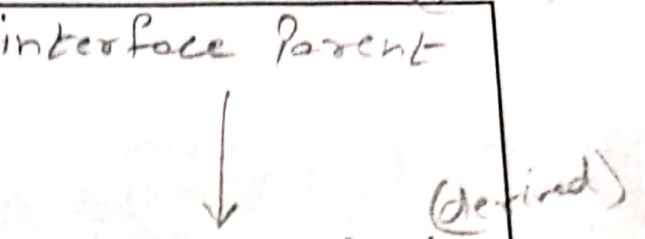
Extending Interfaces

An interface can extend another interface, similarly to the way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface. Syntax is also same as for class inheritance.

```
interface childinterface extends parentinterface
{
    ...
    ...        // body of child interface
}
```

Example 9.2

```
interface parent
{
    int roll = 1001;
    int marks = 85;
}
interface child extends parent
{
    void display();
}
```



In this example, we have used two constants in **parent** interface and one method in **child** interface. The interface **child** would inherit all the constants into it. A java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, an interface can extend more than one parent interface.

The extends keyword is used once and the parent interfaces are declared in a comma-separated list.

Example 9.3

If the student interface extended both sports and test, it would be declared as :

interface student extends sports, test

Note that an interface cannot extend classes.

9.7 Implementing Interfaces

Interfaces can be used as parent classes or superclasses whose properties are inherited by classes. After an interface has been defined, one or more classes can implement it. When a class implements an interface, it must define (or implement) all the methods of the interface. A class uses the **implements** keyword to implement an interface.

Syntax is :

```
class classname implements interfacename
{
...
...        // body of class
}
```

Example 9.4

```
interface messageprint
{
    void display();
}

class sample implements messageprint

{
    public void display()      // note public access specifier
    {
        System.out.println("Implementing Interface");
    }
}

class interfacedemo

{
    public static void main(String args[])
    {
        sample obj = new sample( );
        obj.display();
    }
}
```

Output :

Implementing Interface

Note that the method **display( )** is declared using the **public** access specifier in the class **sample** whenever we implement a method defined by an interface, it must be implemented as **public** since all members of an interface are **implicitly public.**

9.7.1 Extends and Implements Together

A class can implement more than one interface at a time. But a class can extend only one class. Therefore, a class can extend only one class, but implement many interfaces same time. A general form of implementation is :

```
class classname extends baseclass implements interface1, interface2, ...
{
...
    // body of class
...
}
```

Example 9.5

Following example shows that a class can extend another class while implementing interfaces.

studentdata — roll, age

studentmethod — display()

test (baseclass) → student (desired class)

```
// Interface one
interface studentdata
{
    int roll = 1001;
    int age = 18;
}
interface studentmethod
{
    void display();
}
class test
{
    int sub1, sub2;
    void getmarks (int a, int b)
    {
        sub1 = a;
        sub2 = b;
    }
    void printmarks()
    {
        System.out.println(" Marks of Subject1 = " + sub1);
        System.out.println(" Marks of Subject2 = " + sub2);
    }
}
```

```
class student extends test implements studentdata, studentmethod
{
    public void display( )
    {
        System.out.println(" Roll No. = " + roll);
        System.out.println(" Age = " + age);
    }

    void print( )
    {
        display( );
        printmarks( );
        System.out.println(" Total Marks = " + (sub1+sub2));
    }
}
class demointerface
{
    public static void main (String args[])
    {
        student s = new student( );
        s.getmarks (80, 75);
        s.print();
    }
}
```

Output :

```
Roll No. = 1001
Age = 18
Marks of Subject1 = 80
Marks of Subject2 = 75
Total Marks = 155
```

2 Multiple Inheritance in Java

Java supports multiple inheritance, If a class implements multiple interfaces, or an interface extends multiple interfaces as shown in figure 9.2.

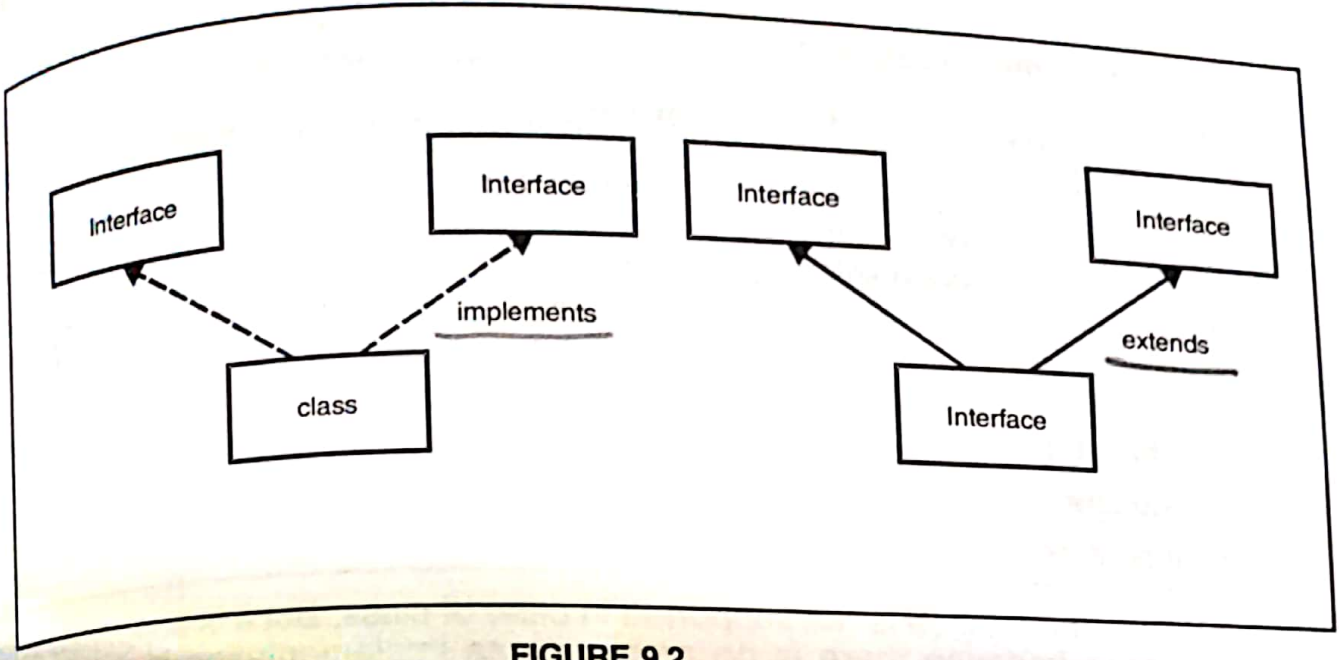


FIGURE 9.2

Previous example is an example of multiple inheritance.

Example 9.6

Very simple example of multiple inheritance is given below :

```
interface first
{
     void print( );
}

interface second
{
     void display( );
}

class midemo implements first, second
{
     public void print( )
     {
          System.out.println("Multiple");
     }
     public void display( )
     {
          System.out.println("Inheritance");
     }
}
```

```
class Multipleinheritance
{
        public static void main (String args[])
        {
                midemo obj = new midemo( );
                obj.print( );
                obj.display( );
        }
}
```

Output :

Multiple
Inheritance

Multiple Inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

Example 9.7

```
interface first
{
        void display( );
}

interface second
{
        void display( );
}

class midemo implements first, second
{
        public void display( )
        {
                System.out.println("Multiple Inheritance");
        }

}

class Multipleinheritance
{
        public static void main (String args[])
        {
                midemo obj = new midemo( );
                obj.display( );
        }
}
```

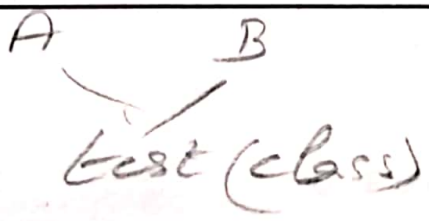
Output :

Multiple Inheritance

In this example, first and second interface have **same method** but its implementation is provided by class **test**, so there is no ambiguity.

Also, two interface can have same constant name.

Example 9.8




```
interface A
{
    int x=20;
}

interface B
{
    int x=500;
}

class test implements A, B
{
    public static void main(String args[])

    {
        System.out.println(x);    // reference to x is ambiguous both variables are x
        System.out.println(A.x); // valid
        System.out.println(B.x); // valid
    }
}
```

define and acc

10 Interfaces verses Abstract Classes

An interface is similar to a class without instance and without method bodies. But abstract classes do allow static method definitions and interfaces don't. So an interface is like an abstract class that must be extended in exactly the manner that its abstract methods specify. The table 9.2 shows the differences between interfaces and abstract classes.

TABLE 9.2

Interface	Abstract Class
1. The **interface** keyword is used to declare a interface.	1. The **abstract** keyword is used to declare a abstract class.
2 Interface can have only abstract methods.	2. Abstract class can have abstract and non-abstract methods.
3. Interface supports multiple inheritance.	3. Abstract class doesn't support multiple inheritace.
4. Interface has only static and final variables.	4. Abstract class can have final, non-final, static and non-static variables.
5. Slow, it requires extra indirection to find corresponding method in the actual class.	5. Fast.
6. Interface can't implement class.	6. Abstract class implements the interface.
7. Interface can't have static methods, main method and constructors.	7. Abstract class can have static methods, main method and constructors.