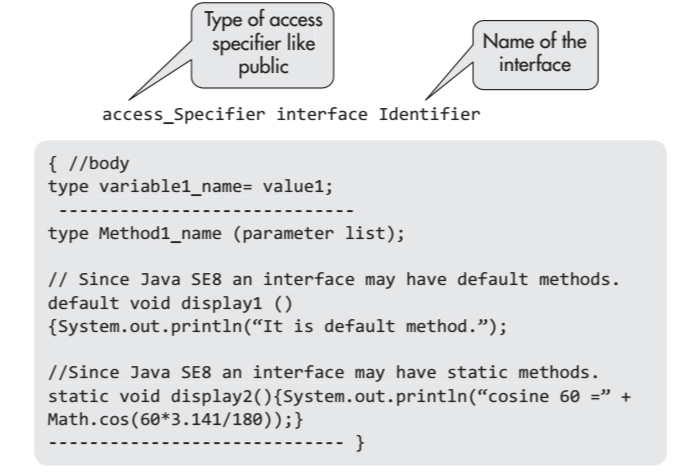
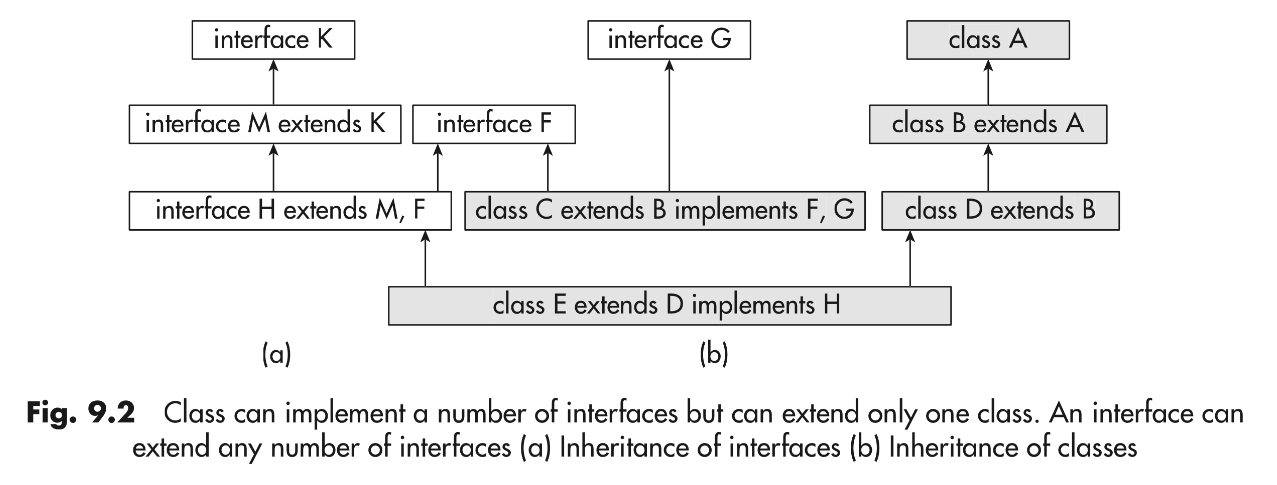
**Declaration of Interface**

Declaration of an interface starts with the access modifier followed by keyword interface which is in turn followed by its name or identifier that is followed by a block of statements; these statements contain declarations of variables and abstract methods. The variables defined in interfaces are implicitly public, static, and final. They are initialized at the time of declaration. The methods declared in an interface are public by default. An illustration of declaration of an interface is given.



The multiple inheritances of classes are not allowed in Java, and therefore, interfaces provide a stopgap arrangement. A class can extend (inherit) another class as well as implement a number of interfaces. For details on inheritanceWith the enhancement of Java SE8, the benefits of multiple inheritances can be realized easily. The interfaces can be extended as well as nested like classes. However, there are differences too. The similarities and dissimilarities of an interface with a class are as follows. Figure 9.2 shows how the interfaces inherit other interfaces and the classes that implement them. A class may implement a number of interfaces besides having a super class.



**Interface Modifiers**

Access modifiers for an interface are generally public or no access modifier is used. By declaring it as public, interface can be used in any package and in any class.

**Members of Interface**

The members of an interface comprise the following:

1. The members declared in the body of the interface.

2. The members inherited from any super interface that it extends.

3. The methods declared in the interface are implicitly public abstract member methods.

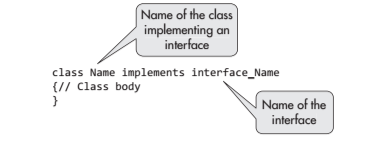
4. The field variables defined in interfaces are implicitly public, static, and final. However, the specification of these modifiers does not create a compile-type error.

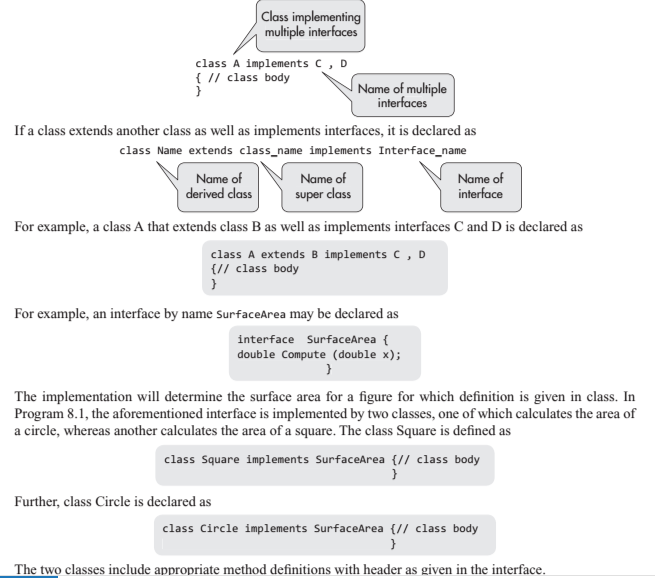
5. The field variables declared in an interface must be initialized; otherwise, compile-type error occurs.

6. Since Java SE8, static and default methods with full definition can also be members of interface.

**Implementation of Interface**

The interfaces are implemented by classes. An illustration of declaration of class that implements an interface.A class implementing an interface must provide implementation for all its methods unless it is an abstract class.





//Illustration of interface to find areas of a square and a circle

interface SurfaceArea { // interface

double Compute(double x);

}// end of interface

class Square implements SurfaceArea// class Square

{

public double Compute(double x) {

return (x \* x);

}

}// end of class Square

class Circle implements SurfaceArea// class Circle

{

public double Compute(double x) {

return (3.141 \* x \* x);

}

}// End of class Circle

class Face {

public static void main(String arg[])

{

Square sqr = new Square();// object of class square

Circle cirl = new Circle (); //class Circle object

SurfaceArea Area;// object reference of interface

// Assigning Square class reference to Area

Area = sqr;

System.out.println("Area of square =" + Area.Compute(10));

// Assigning Circle class reference to Area

Area = cirl;

System.out.println("Area of circle ="+Area.Compute(10));

}

}

**Functional Interfaces**

In Java SE8, a new package java.util.function on functional interfaces has been introduced to serve as the basis for writing Lambda functions. **Functional interfaces are interfaces with one abstract method**. They are also called **SAM or single abstract method** type. However, a functional interface can have more than one static and default methods besides the abstract method and it can override some methods of object class. The programmer may include an annotation, which is given in the following example, in order to lessen the work of complier that anyway will recognize a functional interface.

@FunctionalInterface

it can be helpful in detecting compile time errors. If the functional interface contains more than one abstract method, the compiler will throw an error. As we all know, Java programming language is purely an object-oriented language and an object can call only the methods encapsulated in the class for the object.

Example

@FunctionalInterface

interface sayable{

void say(String msg); // abstract method

// It can contain any number of Object class methods.

int hashCode();

String toString();

boolean equals(Object obj);

}

public class FunctionIn implements sayable{

public void say(String msg){

System.out.println(msg);

}

public static void main(String[] args) {

FunctionIn fie = new FunctionIn();

fie.say("Hello there");

}

}

**Annotations**

it was formally introduced in Javac compiler in Java SE 6. It is a type of metadata that can be integrated with the source code without affecting the running of the program. Comments are also a way of adding information in a source code without affecting the implementation; however, comments are simply neglected at compile time and do not go further. The comments are introduced only for the user/programmer for understanding the program. However, annotations may be retained up to runtime and may be used to instruct the compiler and runtime system to do or not to do certain things.

Since Java SE 8, the annotations may be applied to classes, fields, interfaces, methods, and type declarations like throw clauses. The annotations are no longer simply for metadata inclusion in the program but have become a method for user’s communication with compiler or runtime system.

Example

class XX {

public void display(){System.out.println("This is class XX.");

}

}// end of class XX

class YY extends XX {

@Override public void display()

{

System.out.println("This is class YY.");

}

}// end of class YY

class ZZ extends XX

{ @Override

public void display(){

System.out.println("This is class ZZ.");

}

}// end of class ZZ // below is class with main method public

class OverrideSuper {

public static void main (String Str[])

{

XX objX = new XX();

YY objY = new YY();

ZZ objZ = new ZZ();

objX.display();

objY.display();

objZ.display();

}

}

**Benefits of Using Annotations**

1. It provides useful information to the compiler for detecting errors.

2. The information may also be used for suppressing warnings.

3. Annotations may be used for generating code in xml.

4. Annotation may be retained by another annotation for processing at runtime.

5. It can carry metadata up to runtime and the information may be obtained at runtime.

**Annotation Basics**

An annotation is an interface that declares only methods and is preceded by symbol @. An annotation may be defined with no methods or with one or more methods.

