**Java Abstraction**

**Abstract Class**

* A class that is declared as abstract is known as Abstract class.
* It needs to be extended and its method implemented. It cannot be instantiated.

Key points:

1. Any class that contains one or more abstract methods must also be declared Abstract
2. There can be no objects of an abstract class
3. We cannot declare abstract constructors or abstract static methods
4. Any subclass of an abstract class must either implement all abstract methods in the superclass or be itself declared abstract

Example

**abstract** **class** A {

}

**Abstract method**

A method that is declared as abstract and does not have implementation is known as abstract method.

Example

**abstract** **class** A {

**abstract** **void** printStatus(); //no body and abstract

}

**Example of abstract class**

File: AbstractDemo.java

**abstract** **class** A {

**abstract** **void** callme();

// Concrete methods are allowed in abstract classes

**void** callmetoo() {

System.***out***.println("This is a concrete method");

}

}

**class** B **extends** A {

**void** callme() {

System.***out***.println("B's implementation of callme");

}

}

**class** AbstractDemo {

**public** **static** **void** main(String[] args) {

B b = **new** B();

b.callme();

b.callmetoo();

}

}

**Output:**

B's implementation of callme

This is a concrete method

In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the factory method.

A factory method is the method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

File: TestAbstraction1.java

**abstract** **class** Shape {

**abstract** **void** draw();

}

// In real scenario, implementation is provided by others i.e. unknown by end

// user

**class** Rectangle **extends** Shape {

**void** draw() {

System.***out***.println("drawing rectangle");

}

}

**class** Circle1 **extends** Shape {

**void** draw() {

System.***out***.println("drawing circle");

}

}

// In real scenario, method is called by programmer or user

**class** TestAbstraction1 {

**public** **static** **void** main(String args[]) {

Shape s = **new** Circle1();// In real scenario, object is provided through method e.g. getShape() method

s.draw();

}

}

**Output**

drawing circle

Another example of abstract class in java

File: TestBank.java

**abstract** **class** Bank {

**abstract** **int** getRateOfInterest();

}

**class** SBI **extends** Bank {

**int** getRateOfInterest() {

**return** 7;

}

}

**class** PNB **extends** Bank {

**int** getRateOfInterest() {

**return** 8;

}

}

**class** TestBank {

**public** **static** **void** main(String args[]) {

Bank b;

b = **new** SBI();

System.***out***.println("Rate of Interest is: " + b.getRateOfInterest() + " %");

b = **new** PNB();

System.***out***.println("Rate of Interest is: " + b.getRateOfInterest() + " %");

}

}

**Output:**

Rate of Interest is: 7 %

Rate of Interest is: 8 %

**package** pack1;

**abstract** **class** Mouse { // abstract class can have concrete methods and abstract methods

**abstract** **void** show(); // abstract methods cannot have body

**abstract** **void** eat();

**void** drink() {

System.***out***.println("Mouse drinking");

}

}

**class** Cat **extends** Mouse {

**void** show() {

System.***out***.println("show method in Class Cat");

}

**void** eat() {

System.***out***.println("eat method in Class Cat");

}

**void** drink() {

System.***out***.println("Cat drinking");

}

}

**public** **class** AbstractClassExample {

**public** **static** **void** main(String[] args) {

Cat c1 = **new** Cat();

c1.show();

c1.eat();

c1.drink();

}

}

**Interface in Java**

The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritances in Java.

In other words, you can say that interfaces can have methods and variables but the methods declared in interface contain only method signature, not body.

General form of an Interface

access **interface** name {

return-type method-name1(parameterlist);

return-type method-name2(parameterlist);

…

type varname1 = value;

type varname2 = value;

}

* Here, ‘access’ is either **public** or not used.
* When no access specifier is included then default access results and the interface is available to other members of the package in which it is declared.
* When ‘access’ is declared as public then interface is available to members of all other packages
* Methods that are declared have no bodies and they end with semicolon. ~~There can be no default implementation of any method in the interface~~
* Variables can be declared inside interface. They are implicitly final and static i.e. they cannot be changed by the implementing class. They must also be initialized with a constant value.
* All methods and variables are implicitly public if the interface itself is declared public.

Example

**interface** Callback {

**void** callback(**int** param);

}

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



**Understanding relationship between classes and interfaces**

As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.



In the below example, Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In real scenario, interface is defined by someone but implementation is provided by different implementation providers. And, it is used by someone else. The implementation part is hidden by the user which uses the interface.

File: TestInterface1.java

//Interface declaration: by first user

**interface** Drawable {

**void** draw();

}

// Implementation: by second user

**class** Rectangle **implements** Drawable {

**public** **void** draw() {

System.***out***.println("drawing rectangle");

}

}

**class** Circle **implements** Drawable {

**public** **void** draw() {

System.***out***.println("drawing circle");

}

}

// Using interface: by third user

**class** TestInterface1 {

**public** **static** **void** main(String args[]) {

Drawable d = **new** Circle();

d.draw();

}

}

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



**interface** Printable {

**void** print();

}

**interface** Showable {

**void** show();

}

**class** A7 **implements** Printable, Showable {

**public** **void** print() {

System.***out***.println("Hello");

}

**public** **void** show() {

System.***out***.println("Welcome");

}

}

**class** interface1 {

**public** **static** **void** main(String args[]) {

A7 obj = **new** A7();

obj.print();

obj.show();

}

}

**Output:**

Hello

Welcome

Q) Multiple inheritance is not supported through class in java but it is possible by interface, why?

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

**interface** Printable {

**void** print();

}

**interface** Showable {

**void** print();

}

**class** TestInterface3 **implements** Printable, Showable {

**public** **void** print() {

System.***out***.println("Hello");

}

}

**public** **class** interface1 {

**public** **static** **void** main(String args[]) {

TestInterface3 obj = **new** TestInterface3();

obj.print();

}

}

**Output:**

Hello

**Interface inheritance**

A class implements interface but one interface extends another interface .

**interface** Printable {

**void** print();

}

**interface** Showable **extends** Printable {

**void** show();

}

**class** TestInterface4 **implements** Showable {

**public** **void** print() {

System.***out***.println("Hello");

}

**public** **void** show() {

System.***out***.println("Welcome");

}

}

**public** **class** interface1 {

**public** **static** **void** main(String args[]) {

TestInterface4 obj = **new** TestInterface4();

obj.print();

obj.show();

}

}

**Output:**

Hello

Welcome

Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

File: TestInterfaceDefault.java

**interface** Drawable {

**void** draw();

**default** **void** msg() {

System.***out***.println("default method");

}

}

**class** Rectangle **implements** Drawable {

**public** **void** draw() {

System.***out***.println("drawing rectangle");

}

}

**class** TestInterfaceDefault {

**public** **static** **void** main(String args[]) {

Drawable d = **new** Rectangle();

d.draw();

d.msg();

}

}

Output:

drawing rectangle

default method

Since Java 8, we can have static method in interface. Let's see an example:

File: TestInterfaceStatic.java

**interface** Drawable {

**void** draw();

**static** **int** cube(**int** x) {

**return** x \* x \* x;

}

}

**class** Rectangle **implements** Drawable {

**public** **void** draw() {

System.***out***.println("drawing rectangle");

}

}

**class** TestInterfaceStatic {

**public** **static** **void** main(String args[]) {

Drawable d = **new** Rectangle();

d.draw();

System.***out***.println(Drawable.*cube*(3));

}

}

**Output:**

drawing rectangle

27

**Java Nested Interface**

An interface i.e. declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

**Points to remember for nested interfaces**

* Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
* Nested interfaces are declared static implicitly.

Syntax of nested interface which is declared within the interface

interface interface\_name{

...

interface nested\_interface\_name{

...

}

}

Syntax of nested interface which is declared within the class

class class\_name{

...

interface nested\_interface\_name{

...

}

}

Example of nested interface which is declared within the interface

In this example, we are going to learn how to declare the nested interface and how we can access it.

**interface** Showable {

**void** show();

**interface** Message {

**void** msg();

}

}

**class** TestNestedInterface1 **implements** Showable.Message {

**public** **void** msg() {

System.***out***.println("Hello nested interface");

}

**public** **static** **void** main(String args[]) {

Showable.Message message = **new** TestNestedInterface1();// upcasting here

message.msg();

}

}

**Output:**

hello nested interface

As you can see in the above example, we are acessing the Message interface by its outer interface Showable because it cannot be accessed directly. It is just like almirah inside the room, we cannot access the almirah directly because we must enter the room first. In collection frameword, sun microsystem has provided a nested interface Entry. Entry is the subinterface of Map i.e. accessed by Map.Entry.

Internal code generated by the java compiler for nested interface Message

The java compiler internally creates public and static interface as displayed below:.

public static interface Showable$Message

{

public abstract void msg();

}

Example of nested interface which is declared within the class

Let's see how can we define an interface inside the class and how can we access it.

**class** A {

**interface** Message {

**void** msg();

}

}

**class** TestNestedInterface2 **implements** A.Message {

**public** **void** msg() {

System.***out***.println("Hello nested interface");

}

**public** **static** **void** main(String args[]) {

A.Message message = **new** TestNestedInterface2();// upcasting here

message.msg();

}

}

**Output:**

hello nested interface

**Can we define a class inside the interface?**

Yes, If we define a class inside the interface, java compiler creates a static nested class. Let's see how can we define a class within the interface:

interface M{

class A{}

}

**Difference between abstract class and interface**

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods. Since Java 8, it can have default and static methods also. |
| 2) Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| 3) Abstract class can have final, non-final, static and non-static variables. | Interface has only static and final variables. |
| 4) The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| 5) An abstract class can extend another Java class and implement multiple Java interfaces. | An interface can extend another Java interface only. |
| 6) An abstract class can be extended using keyword ?extends?. | An interface class can be implemented using keyword ?implements?. |
| 7) A Java abstract class can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 8)Example: public abstract class Shape{ public abstract void draw(); } | Example: public interface Drawable{ void draw(); } |

//Creating interface that has 4 methods

**interface** A {

**void** a();// bydefault, public and abstract

**void** b();

**void** c();

**void** d();

}

// Creating abstract class that provides the implementation of one method of A

// interface

**abstract** **class** B **implements** A {

**public** **void** c() {

System.***out***.println("I am C");

}

}

// Creating subclass of abstract class, now we need to provide the

// implementation of rest of the methods

**class** M **extends** B {

**public** **void** a() {

System.***out***.println("I am a");

}

**public** **void** b() {

System.***out***.println("I am b");

}

**public** **void** d() {

System.***out***.println("I am d");

}

}

// Creating a test class that calls the methods of A interface

**class** Test5 {

**public** **static** **void** main(String args[]){

A a=**new** M();

a.a();

a.b();

a.c();

a.d();

}

}