### Abstract class

If a class contain any abstract method then the class is declared as abstract class. An abstract class is never instantiated. It is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method.

**Syntax :**

abstract class class\_name { }

#### Abstract method

Method that are declared without any body within an abstract class are called **abstract method**. The method body will be defined by its subclass. Abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.

**Syntax :**

abstract return\_type function\_name (); //No definition

#### Example of Abstract class

abstract class A

{

abstract void callme();

}

class B extends A

{

void callme()

{

System.out.println("this is callme.");

}

public static void main(String[] args)

{

B b = new B();

b.callme();

}

}

this is callme.

#### Abstract class with concrete(normal) method.

Abstract classes can also have normal methods with definitions, along with abstract methods.

abstract class A

{

abstract void callme();

public void normal()

{

System.out.println("this is concrete method");

}

}

class B extends A

{

void callme()

{

System.out.println("this is callme.");

}

public static void main(String[] args)

{

B b = new B();

b.callme();

b.normal();

}

}

this is callme. this is concrete method.

#### Points to Remember

1. Abstract classes are not Interfaces. They are different, we will study this when we will study Interfaces.
2. An abstract class may or may not have an abstract method. But if any class has even a single abstract method, then it must be declared abstract.
3. Abstract classes can have Constructors, Member variables and Normal methods.
4. Abstract classes are never instantiated.
5. When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

#### Abstraction using abstract class

Abstraction is an important feature of OOPS. It means hiding complexity. Abstract class is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method. Lets see how abstract class is used to provide abstraction.

abstract class Vehicle

{

public abstract void engine();

}

public class Car extends Vehicle {

public void engine()

{

System.out.println("Car engine");

//car engine implementation

}

public static void main(String[] args)

{

Vehicle v = new Car();

v.engine();

}

}

Car

engine

Here by casting instance of **Car** type to **Vehicle** reference, we are hiding the complexity of **Car** type under **Vechicle**. Now the **Vehicle** reference can be used to provide the implementation but it will hide the actual implementation process.

#### When to use Abstract Methods & Abstract Class?

Abstract methods are usually declared where two or more subclasses are expected to do a similar thing in different ways through different implementations. These subclasses extend the same Abstract class and provide different implementations for the abstract methods.

Abstract classes are used to define generic types of behaviors at the top of an object-oriented programming class hierarchy, and use its subclasses to provide implementation details of the abstract class.

### Interface

Interface is a pure abstract class.They are syntactically similar to classes, but you cannot create instance of an **Interface** and their methods are declared without any body. Interface is used to achieve complete **abstraction** in Java. When you create an interface it defines what a class can do without saying anything about how the class will do it.

**Syntax :**

interface interface\_name { }

#### Example of Interface

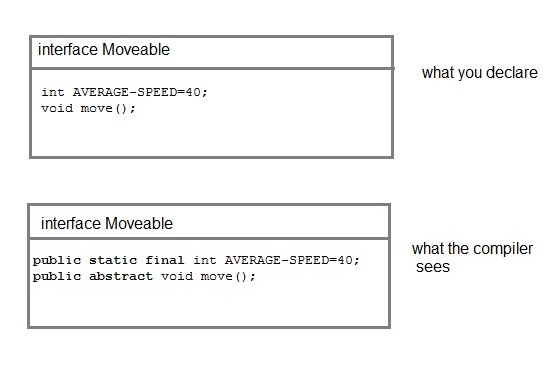
interface Moveable

{

int AVERAGE-SPEED=40;

void move();

}



**NOTE :** Compiler automatically converts methods of Interface as public and abstract, and the data members as public, static and final by default.

#### Rules for using Interface

* Methods inside Interface must not be static, final, native or strictfp.
* All variables declared inside interface are implicitly public static final variables(constants).
* All methods declared inside Java Interfaces are implicitly public and abstract, even if you don't use public or abstract keyword.
* Interface can extend one or more other interface.
* Interface cannot implement a class.
* Interface can be nested inside another interface.

#### Example of Interface implementation

interface Moveable

{

int AVG-SPEED = 40;

void move();

}

class Vehicle implements Moveable

{

public void move()

{

System .out. print in ("Average speed is"+AVG-SPEED");

}

public static void main (String[] arg)

{

Vehicle vc = new Vehicle();

vc.move();

}

}

Average speed is 40.

#### Interfaces supports Multiple Inheritance

Though classes in java doesn't suppost multiple inheritance, but a class can implement more than one interface.

interface Moveable

{

boolean isMoveable();

}

interface Rollable

{

boolean isRollable

}

class Tyre implements Moveable, Rollable

{

int width;

boolean isMoveable()

{

return true;

}

boolean isRollable()

{

return true;

}

public static void main(String args[])

{

Tyre tr=new Tyre();

System.out.println(tr.isMoveable());

System.out.println(tr.isRollable());

}

}

true

true