Abstract:

Abstract is the modifier applicable for classes and methods but not for variables.

Abstract method

- A method that is declared as abstract and does not have implementation (body) is known as abstract method. i. e. abstract method can have only method declaration but not implementation. Hence every abstract method declaration should compulsory ends with ;(semicolon).
- Child class is responsible to provide implementation for parent class abstract methods.
- By declaring abstract methods in parent class we define guidelines to the child classes which describes the methods those are to be compulsory implemented by child class.

Example abstract method

1. abstract void display();   //only declaration, no body defined.
2. abstract void display() //wrong

   {

   }

Abstract class in Java

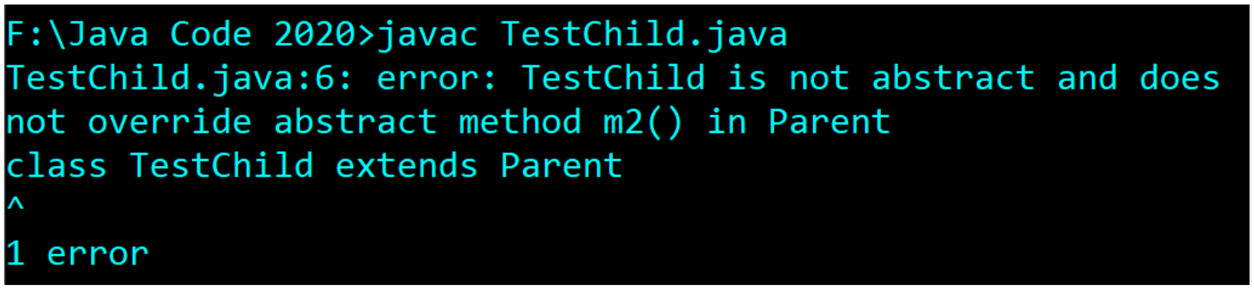
- A class that is declared with abstract keyword is known as abstract class in java.
- It can have abstract method (method without body) and non-abstract methods (method with body).
- It needs to be extended and its method should be implemented.
- It cannot be instantiated.

Example abstract class

1. abstract class A{}

Example:

```
abstract class Parent
{
public abstract void m1();
public abstract void m2();
}
class TestChild extends Parent
{
public  void m1(){    }
}
```



```
F:\Java Code 2020>javac TestChild.java
TestChild.java:6: error: TestChild is not abstract and does
not override abstract method m2() in Parent
class TestChild extends Parent
^
1 error
```

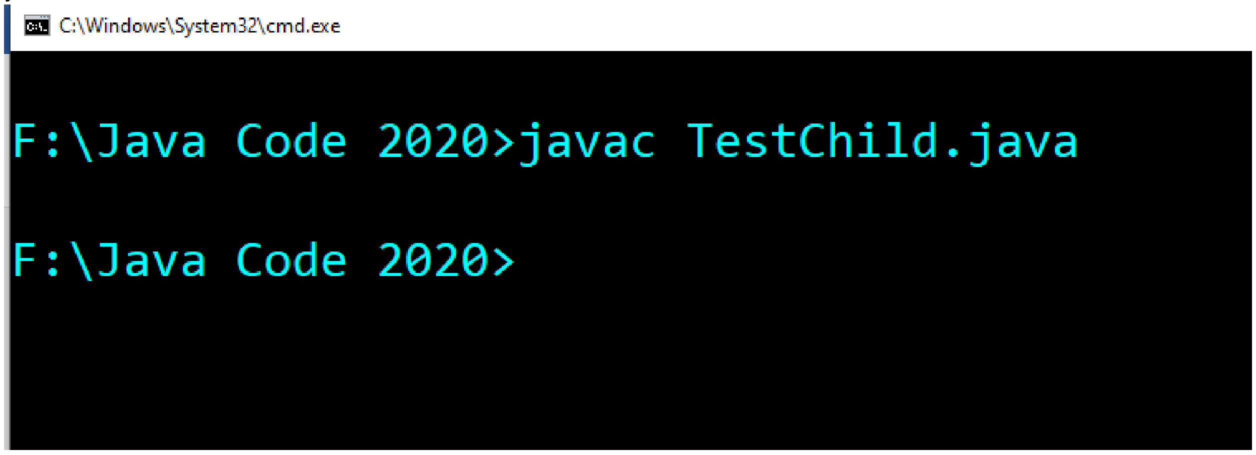
We can handle this error either by declaring child class as abstract (case 1) or by providing implementation for m2() (case 2).

case 1:

```
abstract class Parent
{
public abstract void m1();
public abstract void m2();
}
abstract class TestChild extends Parent
{
public  void m1(){}
}
```

case 2

```
abstract class Parent
{
public abstract void m1();
public abstract void m2();
}
 class TestChild extends Parent
{
public  void m1(){}
public  void m2(){}
}
```



C:\Windows\System32\cmd.exe

```
F:\Java Code 2020>javac TestChild.java

F:\Java Code 2020>
```

Example of abstract class that has abstract method

In this example, Bike is the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike
{
abstract void run();  //abstract method
void speed(){// non abstract method
}
}

class Honda extends Bike
{
void run()
{
System.out.println("Overridden Method run()");
}
public static void main(String args[])
{
 //Bike obj = new Honda();
Honda obj= new Honda()
 obj.run();
}

}
```

Output:

```
Overridden Method run()
```

Note: The use of abstract methods, abstract class and interfaces are recommended and it is always good programming practice.

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending email, you just type the text and send the message. You don't know the internal processing about the message delivery.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

INTERFACE IN JAVA

An **interface** is a blueprint of a class. The interface is **a mechanism to achieve fully abstraction** in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java. It cannot be instantiated just like abstract class.

WHY USE INTERFACE?

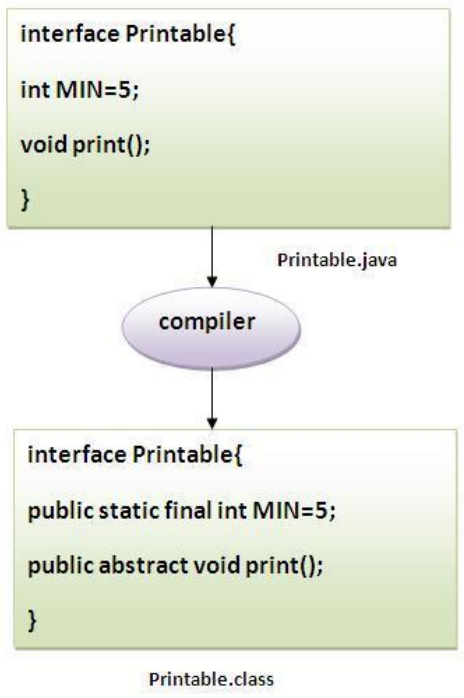
There are following reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.

Interface fields and Methods:

In other words, Interface fields are public, static and final by default, and methods are public and abstract.

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

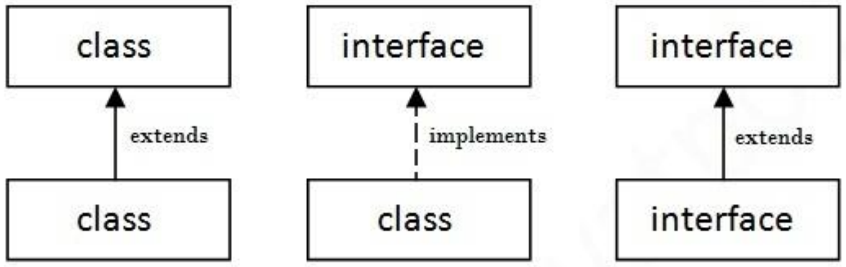


```
interface Printable{

int MIN=5;

void print();

}
```

```
interface Printable{

public static final int MIN=5;

public abstract void print();

}
```

Understanding relationship between classes and interfaces

- A class extends another class.
- An interface extends another interface
- but a **class implements an interface**.



extends Vs implements

A class can extends only one class at a time.

class A extends B ✓

class A extends B, C ☒

A class can implement any number of interfaces at a time.

class A implements B, C ✓

A class can extend a class and implement any number of interfaces simultaneously.

class A extends B implements C,D ✓

class A implements B,C extends D ☒

An interface can extends any number of interfaces at a time.

class A implements B,C ✓

Which of the following is true??

1. A class can extend any number of class simultaneously.
2. A class implement only one interface at a time.
3. A class extend a class OR implement an interface but not both simultaneously.
4. An interface can implement any number of classes.
5. An interface can extend only one interface at a time.
6. None of the above.

Consider the expressions:

X extends Y

Which of the following property is true?

1. Both should be classes
2. Both should be interfaces
3. No restriction
4. Both can be either classes or interfaces.

X extends Y,Z

1. X,Y,Z should be classes.
2. X should be class and Y,Z should be interfaces.
3. X,Y,Z should be interfaces.

X extends Y implements Z

X,Y should be classes,Z should be interface ✓

X implement Y extends Z   -Compile Time Error

EXAMPLE:A CLASS IMPLEMENTS ONE INTERFACE

```
interface printable{
void print();  //by default public and abstract
}

class A implements printable{
public void print()//overridden method
{
System.out.println("Hello");
}

public static void main(String args[]){
//printable obj = new printable ();//        ✗
//A a=new A();                               ✓
printable obj = new A ();//                  ✓
obj.print();
 }
}
```

```
Output:Hello
```

EXAMPLE: TWO CLASSES IMPLEMENTS ONE INTERFACE

```
interface Drawable{
void draw();
}

class Rectangle implements Drawable{
public void draw()//overridden
{
System.out.println("drawing rectangle");
}
}

class Circle implements Drawable
{
public void draw()//overridden
{
System.out.println("drawing circle");
}
}

class TestInterface
```

```
{
public static void main(String args[])
{
Drawable d=new Circle();
Drawable e =new Rectangle();

d.draw();
e.draw();


}
}
```

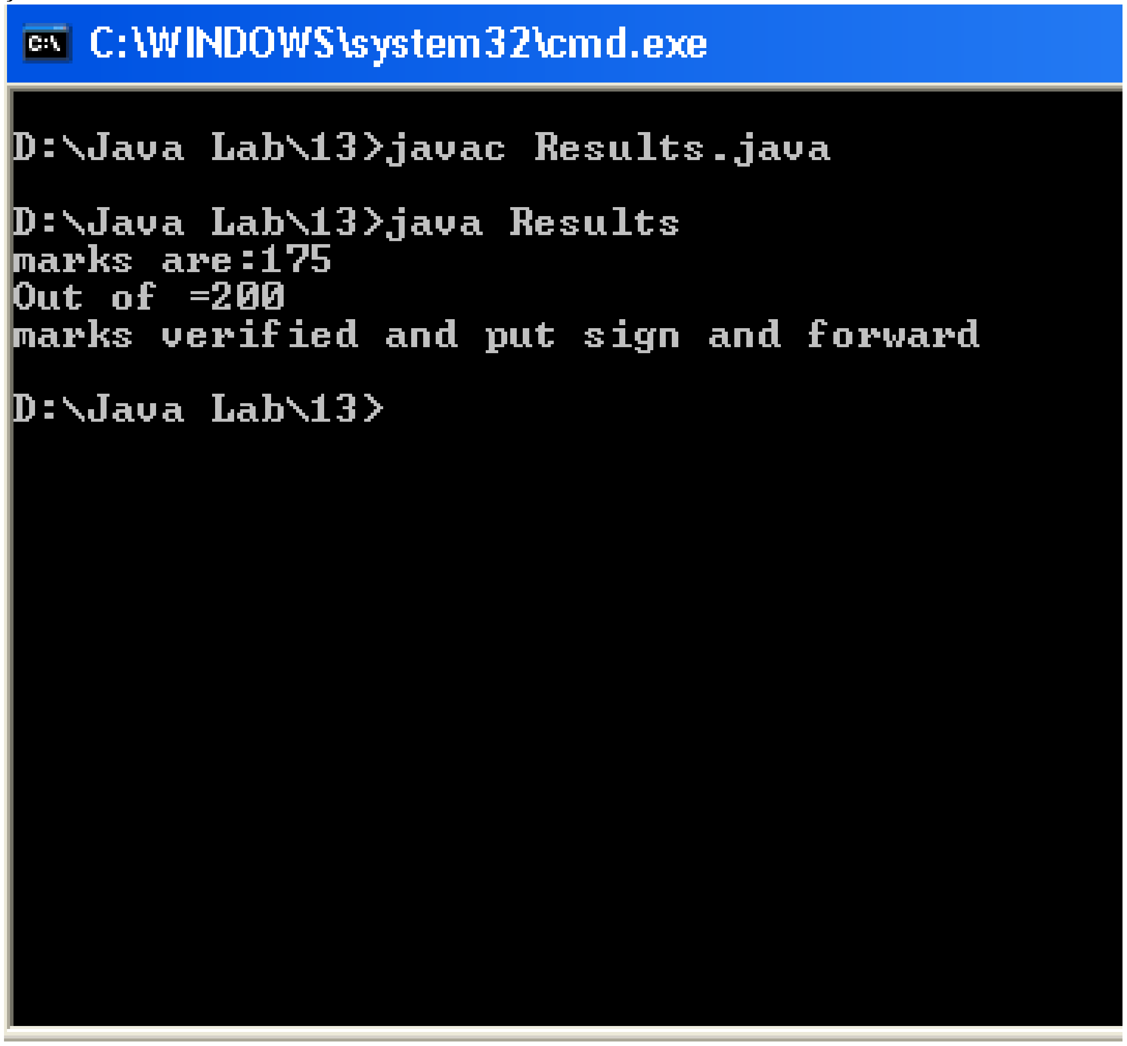
Output:

```
drawing circle

drawing rectangle
```

EXAMPLE: A CLASS EXTENDS ONE CLASS AND IMPLEMENTS ONE INTERFACE (MULTIPLE INHERITANCE)

```
class Teacher
{
int marks;
void setMark(int m)
{
marks=m;
}
void getMark()
{
System.out.println("marks are:"+marks);
}
}

interface Hod
{
int total=200;
void putSign();
}

class Results extends Teacher implements Hod
{
public void putSign()
{
```

```
System.out.println("marks verified and put sign and forward");
}

void display()
{
System.out.println("Out of ="+total);
}

public static void main(String args[])
{

Results r=new Results();
r.setMark(175);
r.getMark();
r.display();
r.putSign();
}
}
```



```
C:\WINDOWS\system32\cmd.exe

D:\Java Lab\13>javac Results.java

D:\Java Lab\13>java Results
marks are:175
Out of =200
marks verified and put sign and forward

D:\Java Lab\13>
```

//A Class implements multiple interfaces(Multiple inheritance)

```java
interface Printable{
void print();
}

interface Showable{
void show();
}

class A implements Printable,Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A obj = new A();
obj.print();
obj.show();
 }
}
```

```
Output:Hello
       Welcome
```

//No ambiguity in multiple inheritance.

```java
interface Printable{
int print();
}

interface Showable{
void print();
}

class A implements Printable,Showable{

public void print(){System.out.println("Hello");}

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

```
Output:Hello
```

MULTILEVEL INHERITANCE

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

interface Showable extends Printable{
void show();
}

class A implements Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A obj = new A();
obj.print();
obj.show();
 }
}
```

```
Output:Hello
       Welcome
```

Assignment Questions:

Q.4 Differentiate concrete class, abstract class and interface.

Q.5 Is it possible a java class can implement any no of Interfaces simultaneously? Justify with example.

Q. 6 Inside interface every method should be abstract where as in abstract class, also we can take only abstract methods then what is the need of interface?

Q.7 Can we write constructor in interface. Justify with example.

Q.8 Inside abstract class we can take constructor but we can’t create object of abstract class then what is the need of constructor inside abstract class.