

## **Abstract:**

The abstract is the modifier applicable for **classes and methods** but not for variables.

### **Abstract method**

- A method declared as abstract and did not have an implementation (body) is known as an abstract method. i.e. abstract method can have only method declaration but not implementation. Hence every abstract method declaration should compulsorily end with ;(semicolon).
- The child class is responsible for providing the implementation of parent class abstract methods.
- By declaring abstract methods in parent class, we define guidelines for the child classes, which describe the methods to be compulsorily implemented by the child class.

```
abstract void addData(int x,int y);
```

### **Example abstract method**

1. `abstract void display(); //valid-only declaration, nobody defined.`
2. `abstract void display() { } //invalid`

## **Abstract class in Java**

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

### **Example abstract class**

1. `abstract class A{ }`

### Case 1:

```
1
2 abstract class Upes {
3     public static void main(String args[])
4     {
5         // Creating object of class inside main() method
6         Upes u =new Upes () ;
7     }
8 }
```

### Output:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.22000.778]
(c) Microsoft Corporation. All rights reserved.

D:\1 UPES Data aug 2022\Java CSF\abstract>javac Upes.java
Upes.java:6: error: Upes is abstract; cannot be instantiated
        Upes u =new Upes();
                   ^
1 error
```

### Case 2:

```
1
2 class Upes
3 { // Method of this class
4     public void methodOne () ;
5 }
```

### Output:

```
D:\1 UPES Data aug 2022\Java CSF\abstract>javac Upes.java
Upes.java:4: error: missing method body, or declare abstract
    public void methodOne();
           ^
1 error
```

### Case 3:

```
class Upes
{    /// Method of this class
    public abstract void method1() {}
}
```

### Output:

```
D:\1 UPES Data aug 2022\Java CSF\abstract>javac Upes.java
Upes.java:2: error: Upes is not abstract and does not override
abstract method method1() in Upes
class Upes
^
Upes.java:4: error: abstract methods cannot have a body
    public abstract void method1() {}
                        ^
2 errors
```

### Case 4:

```
class Upes
{    // Method of this class
    public abstract void method1();
}
```

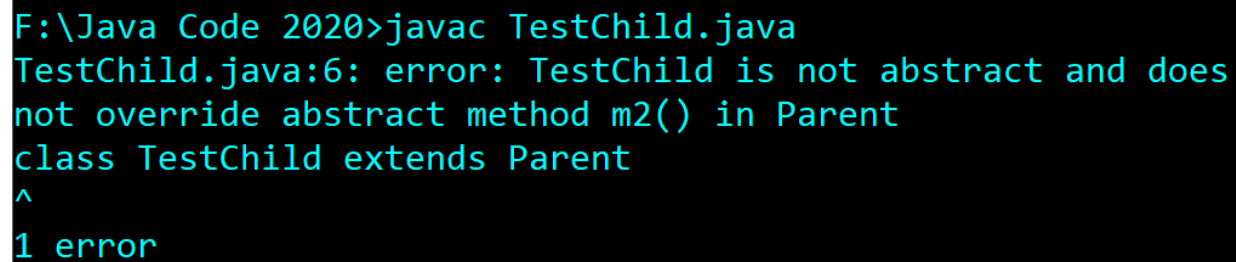
### Output:

```
D:\1 UPES Data aug 2022\Java CSF\abstract>javac Upes.java
Upes.java:2: error: Upes is not abstract and does not override
abstract method method1() in Upes
class Upes
^
1 error
```

Example:

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

**Output:**

A screenshot of a terminal window with a black background and cyan text. The text shows a command to compile a Java file, followed by an error message indicating that the child class is not abstract and does not override the abstract method m2().

```
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 the child class as abstract (case 1) or by providing an 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 an abstract class that has an abstract method

```
// Abstract class  
abstract class Animal {  
    // Abstract method  
    abstract void makeSound();  
  
    // Non-abstract method  
    void sleep() {  
        System.out.println("Sleeping...");  
    }  
}  
  
// Subclass extending the abstract class  
class Dog extends Animal {  
    // Implementing the abstract method  
    void makeSound() {  
        System.out.println("Bark! Bark!");  
    }  
  
    public static void main(String args[]) {  
        // Animal obj = new Animal(); // Compile-time error (Animal is an abstract class)  
        Dog obj = new Dog();  
        obj.makeSound();  
        obj.sleep();  
    }  
}
```

### Output:

```
Bark! Bark!  
Sleeping...
```

## Key Observations About Abstract Classes

- An instance of an abstract class cannot be created.
- Constructors are allowed in abstract classes.
- An abstract class can exist without abstract methods.
- An abstract method cannot be final because a final method cannot be overridden.
- Static methods are allowed in an abstract class.
- The abstract keyword can be used for both outer and inner classes.
- If a class contains at least one abstract method, the class must be declared as abstract.
- If a child class does not implement all abstract methods, it must also be declared as abstract.

### Example 1: Abstract Class with an Abstract Method

```
// Abstract class
abstract class Appliance {
    // Abstract method
    abstract void operate();
}

// Subclass providing implementation
class WashingMachine extends Appliance {
    void operate() {
        System.out.println("Washing clothes...");
    }

    public static void main(String[] args) {
        Appliance obj = new WashingMachine();
        obj.operate();
    }
}

Output:
Washing clothes...
```

### Example 2: Abstract Class with Constructor and Methods

Abstract classes can have constructors and non-abstract methods. The following example demonstrates this:

```
// Abstract class with a constructor and methods
abstract class Vehicle {
    Vehicle() {
        System.out.println("Vehicle is being created");
    }

    // Abstract method
    abstract void start();

    // Non-abstract method
    void stop() {
```

```

        System.out.println("Vehicle stopped");
    }
}

// Subclass implementing abstract method
class Car extends Vehicle {
    void start() {
        System.out.println("Car started");
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        myCar.start();
        myCar.stop();
    }
}

```

**Output:**

Vehicle is being created  
 Car started  
 Vehicle stopped

### Example 3: Abstract Class Without Abstract Methods

An abstract class can exist without any abstract methods. It is useful when we want to restrict instantiation but allow inheritance.

```

abstract class Account {
    void showAccountType() {
        System.out.println("This is a bank account");
    }
}

// Subclass inheriting Account class
class SavingsAccount extends Account {
    void deposit() {
        System.out.println("Money deposited in savings account");
    }
}

public class Main {
    public static void main(String[] args) {
        SavingsAccount acc = new SavingsAccount();
        acc.showAccountType();
        acc.deposit();
    }
}

```

**Output:**

This is a bank account  
 Money deposited in savings account

#### Example 4: Abstract Classes Can Have Final Methods

An abstract class can have final methods, but abstract methods cannot be final.

```
abstract class Computer {
    // Final method cannot be overridden
    final void displayInfo() {
        System.out.println("This is a computer");
    }
}

// Subclass
class Laptop extends Computer {
    // Cannot override displayInfo() because it is final
}
```

```
public class Main {
    public static void main(String[] args) {
        Laptop myLaptop = new Laptop();
        myLaptop.displayInfo();
    }
}
```

Output:

This is a computer

#### Example 5: Abstract Class with Static Methods

Static methods can be defined in an abstract class, and they can be called independently without an object.

```
abstract class MathOperations {
    static void add(int a, int b) {
        System.out.println("Sum: " + (a + b));
    }
}

public class Main {
    public static void main(String[] args) {
        // Calling static method without creating an object
        MathOperations.add(10, 20);
    }
}
```

Output:

Sum: 30

#### Example 6: Abstract Class for Outer and Inner Classes

Abstract classes can be applied to both outer and inner classes.

```
abstract class OuterClass {
    // Abstract inner class
    abstract class InnerClass {
        abstract void message();
    }
}
```



```

}
// Subclass implementing inner abstract class
class SubClass extends OuterClass {
    class ConcreteInner extends InnerClass {
        void message() {
            System.out.println("Inner abstract class implementation");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        SubClass outer = new SubClass();
        SubClass.ConcreteInner inner = outer.new ConcreteInner();
        inner.message();
    }
}
Output:
Inner abstract class implementation

```

### Example 7: Child Class Declared as Abstract if Methods Are Not Implemented

If a child class does not implement all abstract methods of the parent class, it must be declared as abstract.

```

abstract class Animal {
    abstract void sound();
    abstract void eat();
}
// First child class - does not implement all methods, so it must be abstract
abstract class Mammal extends Animal {
    void sound() {
        System.out.println("Mammals make different sounds");
    }
}
// Second child class - implements all remaining methods
class Dog extends Mammal {
    void eat() {
        System.out.println("Dog eats bones");
    }
}
public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.sound();
        myDog.eat();
    }
}

```

**Output:**

Mammals make different sounds

Dog eats bones

## Abstraction in Java

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user. Another way is that it shows only important things to the user and hides the internal details. For example, when sending an email, you type the text and send the message. You don't know how to process the message delivery internally.

### Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

### Examples:

```
abstract class A—0%
{
m1(){stmts.....}—0%
m2(){stmts.....}—0%
m3() {stmts.....}--0%
m4() {stmts.....}--0%
m5(){stmts.....}--0%
}
```

```
abstract class A—60%
{
m1(){stmts.....}—0%
m2(){stmts.....}—0%
abstract m3();--20%
abstract m4();--20%
abstract m5();--20%
}
```

```
abstract class A—100%
{
abstract m1();—20%
abstract m2();—20%
abstract m3();--20%
abstract m4();--20%
abstract m5();--20%
}
```

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