

www.kodnest.com

Abstraction:-

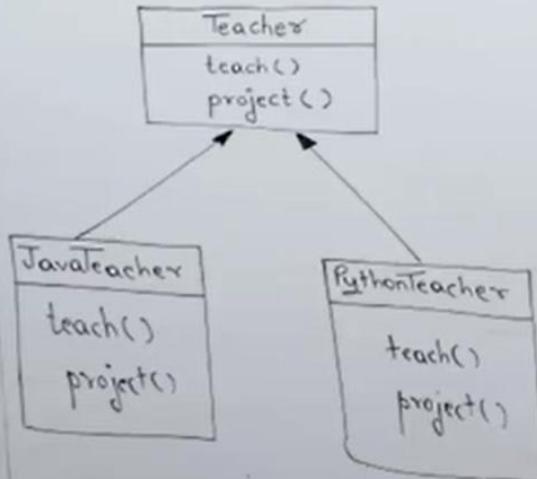
It is the process of hiding irrelevant details. It can be achieved using abstract classes or interfaces.

- Abstraction using abstract classes.

S1

abstract

keyword is used for creating an abstract class.



```

abstract class Teacher
{
    abstract void teach();
    abstract void project();
}

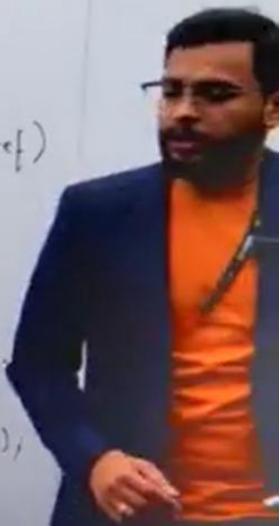
class JavaTeacher extends Teacher
{
    void teach()
    {
        System.out.println("JT is teaching");
    }
    void project()
    {
        System.out.println("JT is doing project");
    }
}
  
```

```

class PythonTeacher extends Teacher
{
    void teach()
    {
        System.out.println("PT is teaching");
    }
    void project()
    {
        System.out.println("PT is doing project");
    }
}

class TeacherApp
{
    static void teacherBehaviour(Teacher ref)
    {
        ref.teach();
        ref.project();
    }

    public static void main(String[] args)
    {
        JavaTeacher jt = new JavaTeacher();
        PythonTeacher pt = new PythonTeacher();
        teacherBehaviour(jt);
        teacherBehaviour(pt);
    }
}
  
```



It is the process of hiding classes or interface not directly accessible.

It can be achieved using abstract classes.

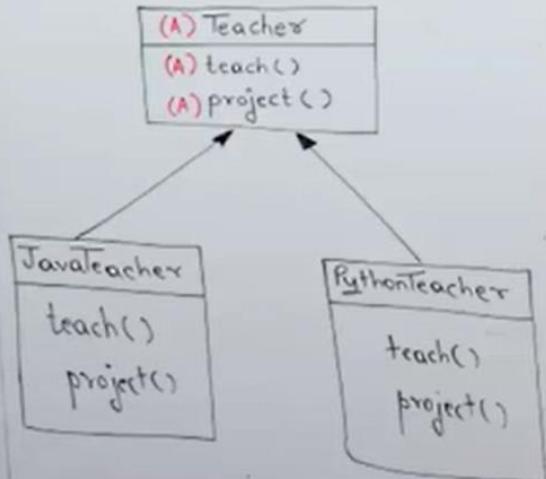
O Abstraction using abstract classes:

SI

abstract keyword is used for creating an abstract class and an abstract method.

abstract methods are such methods which will have the method body also. There is a rule Java that if a class consists of any abstract then the class also must be declared as abstract class.

SE



```

abstract class Teacher
{
    abstract void teach();
    abstract void project();
}

class JavaTeacher extends Teacher
{
    void teach()
    {
        System.out.println("JT is teaching");
    }
    void project()
    {
        System.out.println("JT is doing project");
    }
}
  
```

```

class PythonTeacher extends Teacher
{
    void teach()
    {
        System.out.println("PT is teaching");
    }
    void project()
    {
        System.out.println("PT is doing project");
    }
}

class TeacherApp
{
    static void teacherBehaviour(Teacher ref)
    {
        ref.teach();
        ref.project();
    }

    public static void main(String[] args)
    {
        JavaTeacher jt = new JavaTeacher();
        PythonTeacher pt = new PythonTeacher();
        teacherBehaviour(jt);
        teacherBehaviour(pt);
    }
}
  
```

Java that if a class consists of any non-abstract methods then the class also must be declared as abstract class.

See

An abstract classes can consist of only abstract combination of abstract and concrete and only concrete meths.

What is an abstract class:- for which we can't create the object:-

93

eclipse-workspace1 - AbstractPrograms/src/Program1.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

*Program1.java X ProgramApp.java

```
1 //PURE ABSTRACT CLASS
2 abstract class Program1 {
3     abstract void disp1();
4     abstract void disp2();
5 }
6
```

*Program2.java X

```
1 //IMPURE ABSTRACT CLASS
2 abstract class Program2 {
3     abstract void disp1();
4     void disp2(){
5         System.out.println("inside di
6     }
7 }
8
```

*Program3.java X

```
1 //IMPURE ABSTRACT CLASS
2 abstract class Program3 {
3     void disp1(){
4         System.out.println("inside di
5     }
6     void disp2(){
7         System.out.println("inside di
8     }
9 }
10
```

KodNest

Activate Windows
Go to Settings to activate Windows.

28 min delay SHSS / Hosur M... Search Smart Insert 2:9 [E] ENG IN 6:48 PM 6/14/2024

Note:- If a child class is inheriting the parent class
and parent class consists of abstract methods
then child class must override and provide the
body for every abstract method.
otherwise the child class should be declared as
abstract.

eclipse-workspace1 - AbstractPrograms/src/ProgramApp.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Program1.java X

```
1 abstract class Program1 {  
2     abstract void disp1();  
3     abstract void disp2();  
4 }  
5
```

Program3.java X

```
1 abstract class Program3  
2     void disp1(){  
3 System.out.println("inside disp1");  
4     }  
5     void disp2(){  
6 System.out.println("inside disp2");  
7     }  
8 }
```

ProgramApp.java X

```
1 class ProgramApp {  
2     public static void main(String[] args) {  
3         Program1 p1 = new Program1(); //Cannot instantiate the class Program1  
4     }  
5         Program3 p3 = new Program3(); //Cannot instantiate the class Program3  
6     }  
7 }  
8 }
```

KodNest

Activate Windows
Go to Settings to activate Windows.

Writable Smart Insert 3:32:91

Finance headline "China slams EU..."

Search

Windows Taskbar icons: File Explorer, Edge, ZM, Mail, Google Chrome, FileZilla, Word, Excel, Powerpoint, etc.

ENG US 7:02 PM 6/14/2024

Note: - child class is inheriting the parent class
and parent class consists of abstract methods
then child class must override and provide the
body for every abstract method.
otherwise the child class should be declared as
abstract.

34

Note: - abstract method cannot be declared as
final and static. (because final methods
cannot be overridden)

35

abstract methods can't be called as final because
abstract's methods body has to be provided by
child classes and final method can't be over
ridden by another.

```
Vehicle.java X
1 abstract class Vehicle {
2     abstract void speed();
3     abstract void seater();
4 }
```

```
Car.java X
1 abstract class Car extends Vehicle {
2     void speed() {
3         System.out.println("Car is moving at 180km/hr");
4     }
5 }
```

```
Bike.java X
1 class Bike extends Vehicle {
2     void speed() {
3         System.out.println("Bike is moving at 120km/hr");
4     }
5     void seater() {
6         System.out.println("Bike is 2 seater");
7     }
8 }
9
```

~~final~~ and static (because final m+ve wka.
override n i k a s k + e.

95

abstract methods can't called as final because
abstract's methods body - has to be provided by the
child class's. and final method' cannot be overridden.
which means both opposite to each other.

#A.M is can't declared as static because
static methods are class level methods

96

A screenshot of a Java IDE interface, likely Eclipse, showing the code for an abstract class named `Vehicle`. The code includes a concrete method `speed()` and three abstract methods: `seater()`, `typeEngine()`, and `out.println("vehicle is moving in speed")`. The code is annotated with comments explaining the restrictions on abstract methods.

```
1 abstract class Vehicle
2 {
3     final void speed()
4     {
5         System.out.println("vehicle is moving in speed");
6     }
7
8     //abstract method cannot be declared as final
9     abstract final void seater();
10
11    //abstract method cannot be declared as static
12    abstract static void typeEngine();
13 }
14 |
```

```
Vehicle.java X
1 abstract class Vehicle
2 {
3     static int x;      //static variables
4     static {          //static block
5         x=10;
6     }
7     static void disp1() { //static methods
8         System.out.println(x);
9     }
10    int y;   //non-static variable
11    {        //non-static block
12        System.out.println("Non-static block");
13    }
14    void disp2() // non-static method
15    {
16        System.out.println("static method");
17        System.out.println(x);
18        System.out.println(y);
19    }
20    Vehicle() //constructor
21    {
22        System.out.println();
23    }

```

Writable

Smart Insert

20 : 30 : 418

```
10     int y; //non-static variable
11     {
12         //non-static block
13         System.out.println("Non-static block");
14     }
15     void disp2() // non-static method
16     {
17         System.out.println("non-static method");
18         System.out.println(x);
19         System.out.println(y);
20     }
21     Vehicle() //constructor
22     {
23         System.out.println();
24     }
25     abstract void disp3();
26 }
```

KodNest

Writable

Smart Insert

26 : 27 {22}

An abstract class can consist.

Static variable

static block

static method.

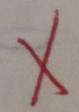
non-static variable

non-static block

non-static method.

constructor.

Abstraction



Abstraction using interfaces :-

S8-9

non-static

constructor.

Abstract class or abstract base class

Abstraction using interfaces :-

SP - 9

Note:- Interface consist of abstract method.
It is another way of achieving abstraction.
When a class connects with an "interface" keyword has to be used which means the "Implemented" class or child class has to provide the body for all the abstract methods otherwise it has to be declared as abstract.

Interfaces can not have concrete methods.

Standardisation-

The screenshot shows a Java development environment with three open files:

- StudentActivities.java**: An interface definition with two abstract methods: `study()` and `practice()`.
- PhysicsStudent.java**: A class that implements `StudentActivities`. It overrides both `study()` and `practice()` methods, each printing a message to `System.out`.
- MathStudent.java**: Another class that implements `StudentActivities`, also overriding `study()` and `practice()` with similar print statements.

```
1 interface StudentActivities {  
2     abstract void study();  
3     abstract void practice();  
4 }  
  
1 class PhysicsStudent implements StudentActivities {  
2     @Override  
3     public void study() {  
4         System.out.println("Physics student is studying");  
5     }  
6     @Override  
7     public void practice() {  
8         System.out.println("Physics student is practicing");  
9     }  
  
1 class MathStudent implements StudentActivities {  
2     @Override  
3     public void study() {  
4         System.out.println("Math student is studying");  
5     }  
6     @Override  
7     public void practice() {  
8         System.out.println("Math student is practicing");  
9     }  
}
```

The screenshot shows a Java development environment with three open files:

- StudentActivities.java**: An interface definition with two abstract methods: `study()` and `practice()`.
- PhysicsStudent.java**: A class that implements `StudentActivities`, providing implementations for both methods.
- StudentApp.java**: The main application class that contains a static method `invokeStudAct` to demonstrate polymorphism, and a `main` method creating instances of `PhysicsStudent` and `MathStudent` and calling the static method.

```
1 interface StudentActivities {  
2     abstract void study();  
3     abstract void practice();  
4 }  
  
1 class PhysicsStudent implements StudentActivities {  
2     // Implementation of StudentActivities methods  
3 }  
  
1 class StudentApp {  
2     static void invokeStudAct(StudentActivities sa) {  
3         sa.study();  
4         sa.practice();  
5     }  
6     public static void main(String[] args) {  
7         PhysicsStudent ps = new PhysicsStudent();  
8         MathStudent ms = new MathStudent();  
9         invokeStudAct(ps);  
10        invokeStudAct(ms);  
11    }  
12 }  
13
```

because of the following reason.

- ① to achieve standardization, which means inside the interface only method declarations with provided in other words interfaces, will tell what to do not how to do.
- ② using interfaces multiple inheritance can be achieved which can not be achieved by abstract classes it is possible using the interface because interface do not have any parent by default.

S₁

- #2. all the methods inside interface are abstract
- # all the methods — by default public
- S₂ - S

The screenshot shows a Java development environment with three open files:

- Calculator.java**: An interface definition with two methods: `void add()` and `void sub()`.
- MyCalculator1.java**: A class implementation of the `Calculator` interface. It contains two methods: `add()` which prints `10+20`, and `sub()` which prints `10-5`.
- CalculatorApp.java**: The main application class. It attempts to create objects of both the interface (`Calculator`) and the concrete class (`MyCalculator1`). The line `Calculator calc1 = new Calculator();` is highlighted in red, indicating a compilation error.

```
Calculator.java
1 interface Calculator {
2     void add();
3     void sub();
4 }

MyCalculator1.java
1 class MyCalculator1 implements Calculator {
2     public void add() {
3         System.out.println(10+20);
4     }
5     public void sub() {
6         System.out.println(10-5);
7     }
8 }

CalculatorApp.java
1 class CalculatorApp {
2     public static void main(String[] args) {
3         Calculator calc1 = new Calculator(); //Cannot create object of interface
4         Calculator calc2; //Reference of interface can be created
5         MyCalculator1 mycalc1 = new MyCalculator1();
6         calc2 = mycalc1;
7         calc2.add();
8         calc2.sub();
9     }
}
```

because of the following reason

- ① to achieve standardization, i- which means inside the interface only method declaration with provided in other words interfaces will tell what to do not how to do
- ② using interfaces multiple inheritance can be achieved which can not be achieved by abstract class it is possible using the interface because interface do not have any parent by default

s₁

- # 2. all the methods inside interface are abstract
- # all the methods — — — by default public

s₂ - s₃

all in interfaces are declared

The screenshot shows a Java development environment with two open code files:

- Calculator.java**: An interface definition with three abstract methods: add(), sub(), and multiply(). The multiply() method includes a comment indicating it corresponds to the public abstract void add() method.

```
1 interface Calculator {
2     void add(); // --> public abstract void add();
3     void sub(); // --> public abstract void sub();
4     public abstract void multiply();
5 }
```


- MyCalculator1.java**: A class implementation that implements the Calculator interface. It provides concrete implementations for all three methods, each printing a result to the console using System.out.println().

```
1 class MyCalculator1 implements Calculator {
2     public void add() {
3         System.out.println(10+20);
4     }
5     public void sub() {
6         System.out.println(10-5);
7     }
8     public void multiply() {
9         System.out.println(10*2);
10    }
11 }
12 
```

The screenshot shows a Java development environment with three open files:

- Calculator.java**: An interface definition.
- MyCalculator1.java**: A class that implements the `Calculator` interface and contains a `change()` method.
- CalculatorApp.java**: A main application class that prints the values of `a` and `b` from `MyCalculator1`.

The code in `Calculator.java`:

```
1 interface Calculator {  
2     int a = 10; // --> public static final int a = 10;  
3     public static final int b = 20;  
4 }
```

The code in `MyCalculator1.java`:

```
1 class MyCalculator1 implements Calculator {  
2     void change()  
3     {  
4         a = 99; //Error, bcz final variables cannot be changed  
5     }  
6 }
```

The code in `CalculatorApp.java`:

```
1 class CalculatorApp {  
2     public static void main(String[] args) {  
3         System.out.println(MyCalculator1.a); //10  
4         System.out.println(MyCalculator1.b); //20  
5     }  
6 }  
7
```

#2. all the methods inside interface are abstract
all the method by default public.
S2 - S3

H3: All the variable in interfaces are declared
as public static and final. ex. S4.

H. If any class is partially implementing interface
ki kuch method over ride ke say hai or kuch ni
the interface (it is not providing the body all abstract
method then the class also have to be an abstract
class.)

The screenshot shows a Java development environment with three code editors:

- Calculator.java**: An interface definition with methods `add()` and `sub()`.
- MyCalculator1.java**: An abstract class that implements the `Calculator` interface. It provides a body for the `add()` method, which prints the sum of 10 and 20. The `sub()` method is declared but not implemented.
- CalculatorApp.java**: A main application class that starts with a single line of code.

```
Calculator.java
1 interface Calculator {
2     void add();
3     void sub();
4 }

MyCalculator1.java
1 abstract class MyCalculator1 implements Calculator {
2     public void add()
3     {
4         System.out.println(10+20);
5     }
6
7     //Not providing the method body for sub()
8 }
9

CalculatorApp.java
1 class CalculatorApp {
```

multiple Inheritance:-

www.kodnest.com

The above example represents multiple inheritance using Interface.

S6

Extends:- Is a relationship (i will override your methods)

Implements:- Promise (i will implement your interface)

Interface can never implement another interface.

The screenshot shows a Java development environment with several open code files:

- Calculator1.java**: An interface definition with a single method `void add();`.
- Calculator2.java**: An interface definition with a single method `void sub();`.
- Calculator3.java**: An interface definition with a single method `void multiply();`.
- MyCalculator1.java**: A class implementation that implements all three interfaces. It contains three methods: `add()`, `sub()`, and `multiply()`. Each method prints a calculation result to `System.out`.
- calculatorApp.java**: A main application class with a `main` method. It creates an instance of `MyCalculator1` and calls its `add()` and `sub()` methods.

The code is color-coded for syntax highlighting, and the IDE interface includes toolbars, status bars, and a project sidebar labeled "KodNest".

```
1 interface Calculator1{  
2     void add();  
3 }  
  
1 interface Calculator2 {  
2     void sub();  
3 }  
  
1 interface Calculator3 {  
2     void multiply();  
3 }  
  
1 class MyCalculator1 implements Calculator1, Calculator2, Calculator3 {  
2     public void add() {  
3         System.out.println(10+20);  
4     }  
5     public void sub(){  
6         System.out.println(10-5);  
7     }  
8     public void multiply(){  
9         System.out.println(10*2);  
10    }  
11 }  
  
1 class CalculatorApp {  
2     public static void main(String[] args) {  
3         MyCalculator1 mycalc1 = new MyCalculator1();  
4         mycalc1.add();  
5         mycalc1.sub();  
6     }  
7 }
```

Extends:- Is a.

Implements:- promise (i will override your method)

interface can never implement another interface
rather it can extend another interface.

If a class is extending the extended
child interface then it has to provide the me-
body for all the methods of child interface.
and parent interface:- Ex. - 97

The screenshot shows a Java development environment with four code files:

- Calculator.java**: An interface definition.
- Calculator2.java**: An interface definition that extends the `Calculator1` interface and adds a `sub()` method.
- MyCalculator1.java**: A class implementation that implements the `Calculator2` interface. It contains two methods: `add()` which prints `10+20`, and `sub()` which prints `10-5`.
- CalculatorApp.java**: A main application class with a `main` method that creates an instance of `MyCalculator1`, calls `add()`, and then `sub()`.

```
Calculator.java
1 interface Calculator1 {
2     void add();
3 }

Calculator2.java
1 interface Calculator2 extends Calculator1 {
2     void sub();
3 }

MyCalculator1.java
1 class MyCalculator1 implements Calculator2 {
2     public void add() {
3         System.out.println(10+20);
4     }
5     public void sub() {
6         System.out.println(10-5);
7     }
8 }

CalculatorApp.java
1 class CalculatorApp {
2     public static void main(String[] args) {
3         MyCalculator1 mycalc1 = new MyCalculator1();
4         mycalc1.add();
5         mycalc1.sub();
6     }
7 }
```

Calculator.java

```
1 interface Calculator1 {  
2     void add();  
3 }
```

*MyCalculator2.java

```
1 class MyCalculator2 {  
2     void sub() {  
3         System.out.println(10-5);  
4     }  
5 }  
6
```

MyCalculator1.java

```
1 class MyCalculator1 extends MyCalculator2 implements Calculator1 {  
2     public void add() {  
3         System.out.println(10+20);  
4     }  
5 }  
6  
7
```

Updates Available

Software updates have been downloaded.
Click to review and install updates.

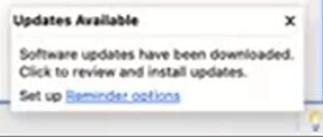
Set up [Reminder options](#)

child interface
body for all the methods
and parent interface: Ex. - 97

A class can extend another class and implement
an interface at the same time.

A marker interface, Tagged interface, empty interface
empty interfaces are called as marker interface
they also consider as tagged interface.
Serializable interface is the marker interface

```
1  
2  
3 //Marker interface , Tagged interface, Empty interface  
4 interface Calculator1  
5 {  
6  
7 }  
8 //Ex:- Serializable interface
```



Writable

Smart Insert

8 : 2 (29)