

59 - 30.

www.kodnest.com

S.9:- complete - getter - getter :-

2# Inheritance :- Is the process. Inheriting  
of properties of One class into  
another class.

In Java, inheritance can be achieve. Extends  
key word.

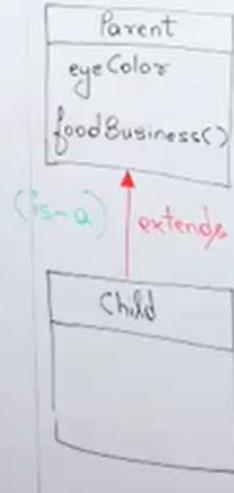
The class which gives properties and behaviour  
is called as parent class and the class which takes  
properties and behaviour is called as child class.

# parent class is also called as Super class and base

# child . class. is also called as derived and Sub

S1, S3

# We can achieve inheritance with extends key word

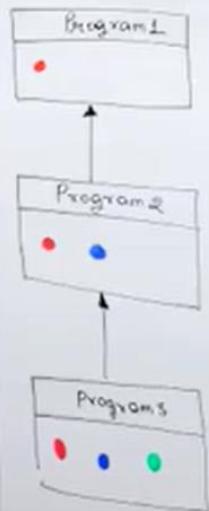


```

class Parent
{
    String eyeColor = "Blue";
    void foodBusiness()
    {
        Sob("Chinese Restro");
    }
}
class Child extends Parent
{
}
  
```

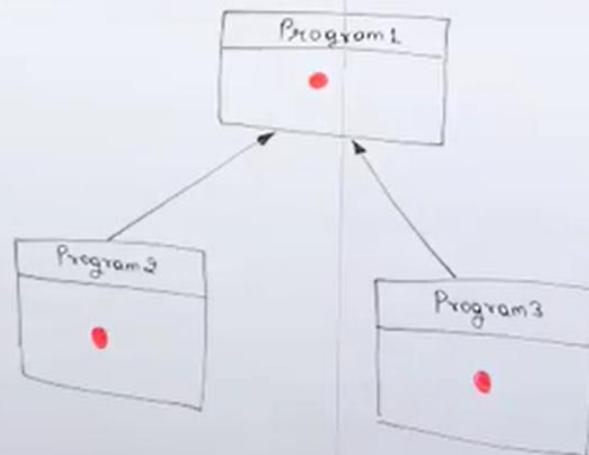
```

class ParentChildApp
{
    public void main(String[] args)
    {
        Parent p = new Parent(),
        Sob(p.eyeColor);
        p.foodBusiness();
        Child c = new Child(),
        Sob(c.eyeColor);
        c.foodBusiness();
    }
}
  
```

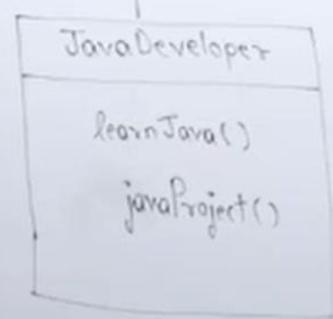
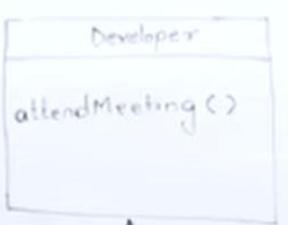


```
class Program1  
{  
    ...  
}  
class Program2 extends Program1  
{  
    ...  
}  
class Program3 extends Program2  
{  
    ...  
}
```

Hierarchical



```
class Program1  
{  
}  
class Program2 extends Program1  
{  
}  
class Program3 extends Program1  
{  
}
```

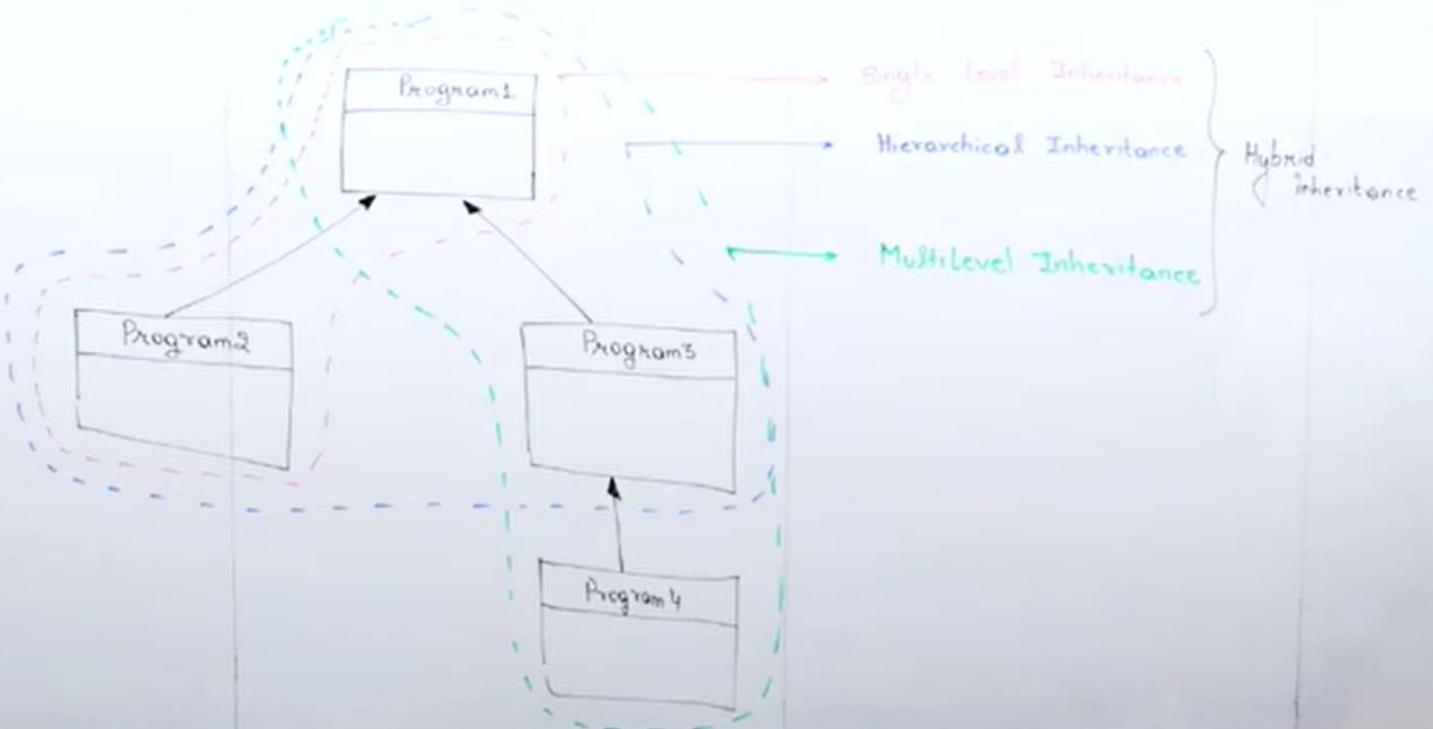


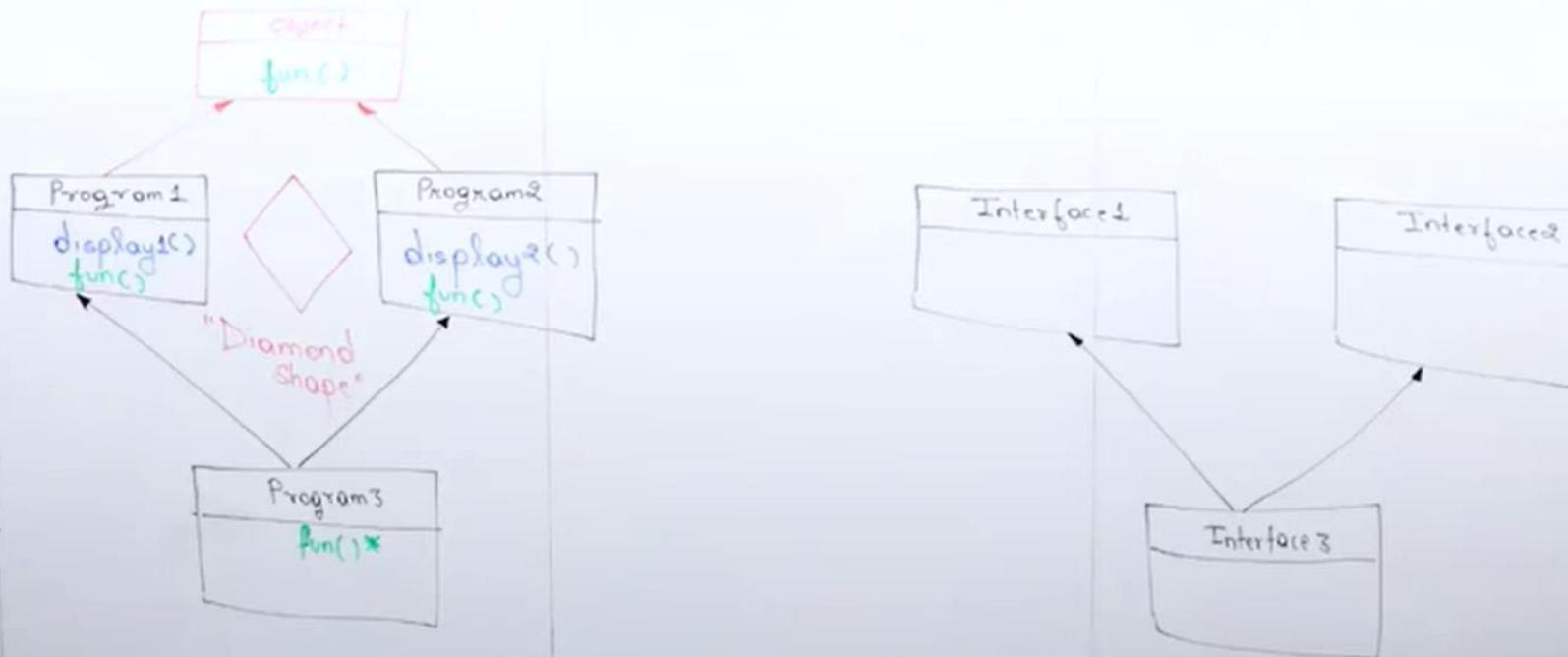
```
class Developer
{
    void attendMeeting()
    {
        System.out.println("Developer is attending the meeting");
    }
}

class JavaDeveloper extends Developer
{
    void learnJava()
    {
        System.out.println("Java developer is learning java");
    }
}
```

```
void javaProject()
{
    System.out.println("Java developer is doing java project");
}

class DeveloperApp
{
    public static void main(String[] args)
    {
        JavaDeveloper jdev = new JavaDeveloper();
        jdev.learnJava();
        jdev.javaProject();
        jdev.attendMeeting();
    }
}
```





## Types of Inheritance.

- ① Single Level Inheritance :- Is such a inheritance in which one parent class is inherited by one child class.  
ex:- As above
- ② Multi Level Inheritance :- Is such a inheritance in which a class will be inherited by its child class and the child class will again inherit by its child class.
- ③ Hierarchical Inheritance :- In which one parent class will be inherited by multiple child classes.
- ④ Hybrid Inheritance :- Which is a combination of two or more type of inheritance.

⑤ **Multiple Inheritance**:- It is the type of inheritance in which a child class will be inheriting from multiple parent classes.

↳ **Advantages**

In Java, multiple inheritance is not allowed using the Java classes because by default every class in Java will have the parent as object hence it will lead to diamond shape problem due to which multiple inheritance is not allowed using class.

But in Java multiple inheritance can be achieved using the interfaces because interfaces do not have any default parent.

⑥ **Cyclic Inheritance**:- In which child class will be inheriting by the parent class and parent class also inheriting the child class.

This type of inheritance is not allowed in Java.

## methods

www.kodnest.com

① Inherited :- So as it is Inherited ho

② Overidden :- Inherited too ne ke haad modified

③ child specific :- ye vo method hai jo agar se child class mai bani Jay.

④ which are

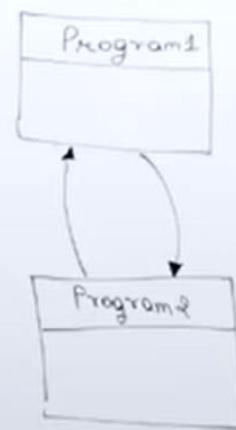
② These are such methods which will be inherited from parent class to child class, but child class will modify the body according to the requirement.

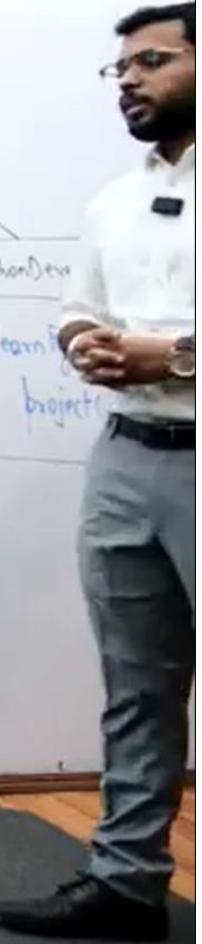
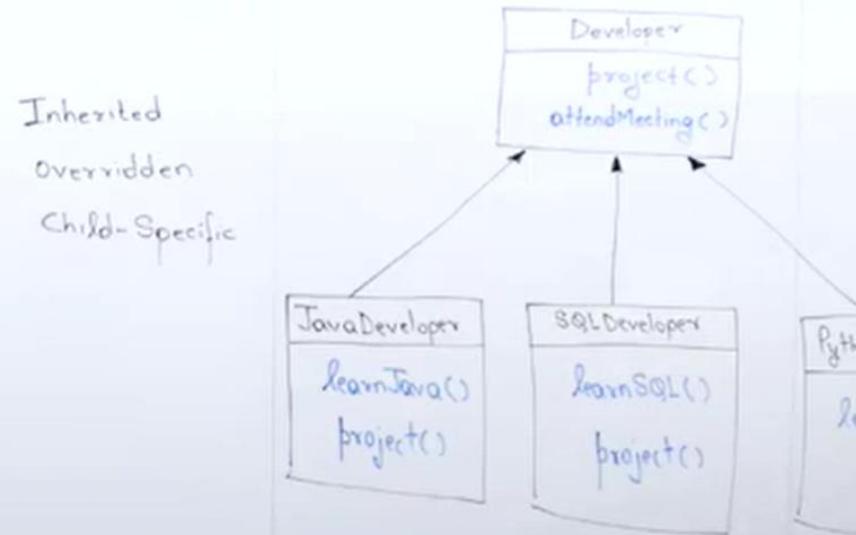
③ C.S :- which are only present only inside the child class and not present ~~inside~~ inside the parent class.

# Constructors :- in

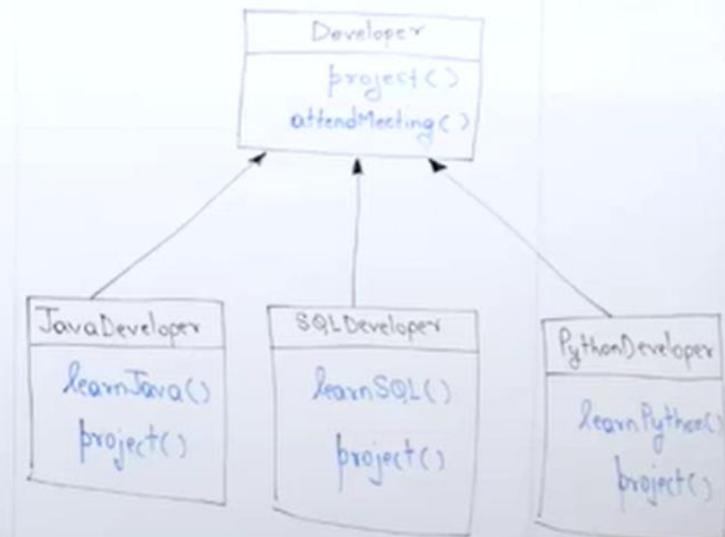
# final keyword :- Can be used with class, method, variable.

① final keyword with class :- is used to prevent inheritance in other words, final classes can not be inherited.





Inherited  
overridden  
Child-Specific



```
class Object  
{  
    Object ()  
}  
}  
  
class Parent extends Object
```

Constructor  
chaining

```
Parent ()  
{  
    super ();  
}
```

```
class Child extends Parent  
{  
    void dispChild()  
    {  
        System.out.println("Inside child disp");  
    }  
  
    Child ()  
    {  
        super ();  
    }  
}
```

JVM

```
class ParentChildApp extends Object  
{  
    public static void main (String [ ] args)  
    {  
        Parent p = new Parent ();  
        p.dispParent();  
  
        Child c = new Child ();  
        c.dispChild();  
        c.dispParent();  
    }  
  
    ParentChildApp()  
    {  
        super ();  
    }  
}
```

## Access modifiers

SSS

private.



we can not access  
in other class

① Public:-  
anywhere.  
Protected  
↓  
diff. package we  
can access.

This variable :- removing shadowing problem

upper variable :- accessing parent class variable

this method :- constructor chaining in same class

super method :- constructor chaining in child class and  
parent class

SS2

final keyword :- we can use with class, method and  
variable.

class with final keyword - Is used to prevent inheritance  
(final classes cannot be inherited)

SS3

final class P1

class P2 extends P1 { } // error

② method:- final keyword with method or is used to prevent method overriding (final methods cannot be overridden (child class can use it as it is))  
S14;

③ Variable:-

① The access modifiers of parent class method and child class overridden method must be same if It is different will get an error.  
S15.

② The access modifiers of parent class method and child class overridden method can be different if visibility and accessibility is increased, but if it is decreased then it is not allowed.  
S16.

③ Private never accessible.

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

- Program1.java**: A class named `Program1` containing a field `a` and a method `disp()` that prints the value of `a`.
- Program2.java**: A class named `Program2` that extends `Program1`. It overrides the `disp()` method to print a different value.
- ProgramApp.java**: A main application class that creates instances of `Program1` and `Program2`, calling their `disp()` methods.

```
Program1.java
2
3 class Program1
4 {
5     int a = 10;
6     void disp() {
7         System.out.println(a); //10
8     }
9 }
10
11
12
13

Program2.java
3 class Program2 extends Program1
4 {
5     int a = 55;
6     void disp()
7     {
8         System.out.println(a); //55
9     }
10 }

ProgramApp.java
1 package com.learn.inheritance.superthis
2
3 public class ProgramApp {
4
5     public static void main(String[]
6
7         Program1 p1 = new Program1();
8         p1.disp();
9
10        Program2 p2 = new Program2();
11        p2.disp();
12
13    }
14
15 }
16
```

```
2  
3  
4 class Program1  
5 {  
6     final void disp()  
7     {  
8         System.out.println("inside program 1 disp");  
9     }  
10 }  
11  
12  
13  
14
```

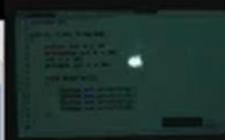
```
ss Program2 extends Program1  
5  
6 @Override  
7 void disp() //--> Error, bcz final cant be overridden  
8 {  
9     System.out.println("inside program2 disp");  
10 }
```

```
21  
22  
23  
24 final keyword  
25  
26 - class (t  
27 - method (t  
28 - variable (t  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43
```

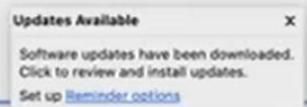
Updates Available

Software updates have been downloaded.  
Click to review and install updates.  
[Set up Reminder options](#)

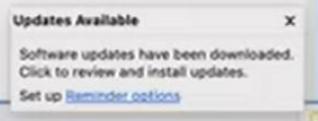
```
1 package p1;
2
3 public class Program1
4 {
5     public int a = 10;
6     protected int b = 20;
7     int c = 30;
8     private int d = 40;
9
10    void display1()
11    {
12        System.out.println(a);
13        System.out.println(b);
14        System.out.println(c);
15        System.out.println(d);
16    }
17 }
18
```



KodNest



```
1 package p1;
2
3 public class Program2 extends Program1
4 {
5     void display2()
6     {
7         System.out.println(a);
8         System.out.println(b);
9         System.out.println(c);
10        System.out.println(d); //error
11    }
12 }
13
14
15
16
17
18
19
```



A screenshot of a Java IDE interface. The main window displays the code for `Program3.java`. The code is as follows:

```
1 package p1;
2 public class Program3
3 {
4     void display3()
5     {
6         Program1 p1 = new Program1();
7         System.out.println(p1.a);
8         System.out.println(p1.b);
9         System.out.println(p1.c);
10        System.out.println(p1.d); //error
11    }
12 }
13
```

The code uses the `Program1` class, which is defined in another file. The `System.out.println(p1.d);` line is commented out with a double slash and labeled as an error. The IDE has a toolbar at the top with various icons, and a status bar at the bottom showing "Writable", "Smart Insert", and the time "3 : 2 : 36". A floating window titled "Updates Available" is visible in the bottom right corner, stating "Software updates have been downloaded. Click to review and install updates." and "Set up [Reminder options](#)".

A screenshot of a Java IDE interface. The main window displays the code for `Program4.java`. The code defines a package `p2`, imports `Program1` from `p1`, and creates a class `Program4` that extends `Program1`. It contains a method `display4()` that prints four variables (`a`, `b`, `c`, `d`) using `System.out.println`. Variables `a` and `b` are correctly resolved, while `c` and `d` are marked with red squiggly underlines, indicating they are undefined. The code ends with a closing brace for the class definition.

```
1 package p2;
2
3 import p1.Program1;
4
5 public class Program4 extends Program1
6 {
7     void display4()
8     {
9         System.out.println(a);
10        System.out.println(b);
11        System.out.println(c); //error
12        System.out.println(d); //error
13    }
14
15 }
16
```

In the bottom right corner of the IDE, there is a small floating window titled "Updates Available". It contains the message: "Software updates have been downloaded. Click to review and install updates." Below the message are two buttons: "Set up Reminder options" and a close button "X".

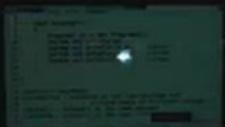
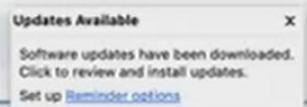
```
1 package p2;
2
3 import p1.Program1;
4
5 public class Program5 {
6
7     void display5()
8     {
9         Program1 p1 = new Program1();
10        System.out.println(p1.a);
11        System.out.println(p1.b);      //error
12        System.out.println(p1.c);      //error
13        System.out.println(p1.d);      //error
14    }
15
16 }
17
18 //public - anywhere
19 //protected - anywhere in the same package and
```

Updates Available

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

Set up [Reminder options](#)

```
6
7     void display5()
8     {
9         Program1 p1 = new Program1();
10        System.out.println(p1.a);
11        System.out.println(p1.b);      //error
12        System.out.println(p1.c);      //error
13        System.out.println(p1.d);      //error
14    }
15
16 }
17
18 //public - anywhere
19 //protected - anywhere in the same package and
20 //                //child class of different package
21 //default - anywhere in the same package
22 //private - anywhere in the same class
23 |
24
```



The screenshot shows a Java development environment with four code editors open:

- Program1.java**:

```
3 class Program1 {  
4     int a = 10;  
5     void disp() {  
6         System.out.println(a); //10  
7     }  
8 }  
9
```
- ProgramApp.java**:

```
3 public class ProgramApp {  
4  
5     public static void main(String[] args) {  
6         Program1 p1 = new Program1();  
7         p1.disp();  
8  
9         Program2 p2 = new Program2();  
10        p2.disp();  
11    }  
12 }  
13  
14 }  
15 }
```
- Program2.java**:

```
3 class Program2 extends Program1 {  
4     int a = 55;  
5     void disp() {  
6         System.out.println(a); //55  
7         System.out.println(super.a); //10  
8     }  
9 }  
10
```
- Program3.java**:

```
3 class Program3 extends Program2 {  
4     int a = 99;  
5     void disp() {  
6         System.out.println(a); //99  
7         System.out.println(super.a); //55  
8         System.out.println(super.super.a); //error  
9     }  
10 }
```

The code editors have syntax highlighting and line numbers. The 'Program3.java' editor has a red error squiggle under the line `System.out.println(super.super.a); //error`. The status bar at the bottom indicates the code is Writable and the current position is 8 : 3 : 169.

Program1.java

```
3 final class Program1
4 {
5
6 }
7
8
9
10
11
12
```

Program2.java

```
3
4 class Program2 extends Program1 {
5
6 }
7
8
9
10
11
12
```

\*ProgramApp.java

```
21
22
23
24 final k
25
26 - class
27 - method
28 - variable
29
30
31
32
33
34
35
36
37
38
```

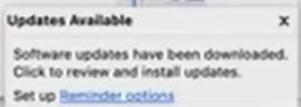
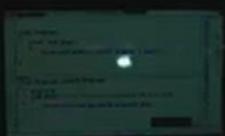
Updates Available

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

Set up [Reminder options](#)

```
2
3
4 class Program1
5 {
6     final void disp()
7     {
8         System.out.println("inside program 1 disp");
9     }
10}
11
12
13
14
```

```
4 class Program2 extends Program1
5 {
6     @Override
7     void disp() //---> Error, bcz final cant be overridden
8     {
9         System.out.println("inside program2 disp");
10    }
11
12}
```



A screenshot of an IDE interface showing a Java file named `Program.java`. The code defines a class `Program1` with a `final int a = 10;` declaration. Inside the `changeValue()` method, there is a line `a=99;` which is highlighted in purple and underlined with a red wavy line, indicating an error. The IDE's status bar at the bottom shows "Writable", "Smart Insert", and the time "24:1:242". A tooltip window titled "Updates Available" is visible in the bottom right corner, stating "Software updates have been downloaded. Click to review and install updates." and "Set up Reminder options".

```
1 package com.kodeinjava;
2
3 class Program1
4 {
5     final int a = 10;
6     void changeValue()
7     {
8         System.out.println(a); //10
9         a=99;    //--> Error ,can't change the value of final variable
10        System.out.println(a); //99
11    }
12 }
13
14
15
16
17
18
19
20
21
22
23
```

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

- Program1.java:** Contains a single class definition:

```
1 class Program1 {  
2     public void display() {  
3         System.out.println("Program1 display");  
4     }  
5 }  
6
```
- Program2.java:** Contains a class definition that extends Program1:

```
1 class Program2 extends Program1 {  
2     @Override  
3     public void display() {  
4         System.out.println("Program2 display");  
5     }  
6     s  
7 }  
8
```
- Program3.java:** Contains a class definition that extends Program1, with a highlighted error in the display method:

```
1 class Program3 extends Program1 {  
2     @Override  
3     void display() { //error  
4         System.out.println("Program3 display");  
5     }  
6 }  
7
```

A tooltip window titled "Updates Available" is displayed in the bottom right corner, containing the message: "Software updates have been downloaded. Click to review and install updates." and a "Set up Reminder options" link.

```
1 class Program1 {  
2     void display() {  
3         System.out.println("Program1 display");  
4     }  
5 }  
6  
1 class Program2 extends Program1 {  
2     @Override  
3     public void display() {  
4         System.out.println("Program2 display");  
5     }  
6 }  
7  
1 class Program3 extends Program1 {  
2     @Override  
3     void display() {  
4         System.out.println("Program3 display");  
5     }  
6 }  
7
```

ProgramAp X

1 public  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11 }  
12

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

- Program1.java**:

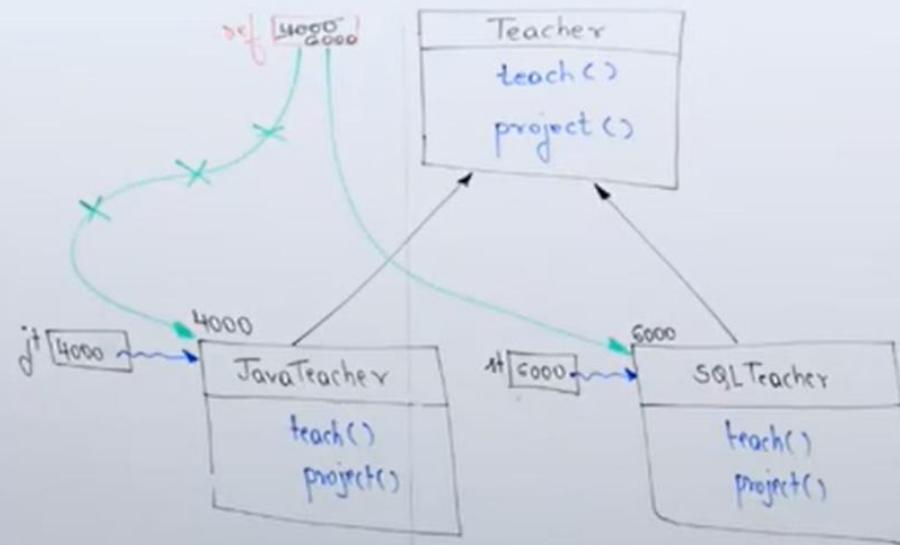
```
1 class Program1 {  
2     protected void display() {  
3         System.out.println("Program1 display");  
4     }  
5 }  
6
```
- Program2.java**:

```
1 class Program2 extends Program1 {  
2     @Override  
3     public void display() { //bcz visibility is increased  
4         System.out.println("Program2 display");  
5     }  
6 }  
7
```
- Program3.java**:

```
1 class Program3 extends Program1 {  
2     @Override  
3     void display() { //error, visibility is decreased  
4         System.out.println("Program3 display");  
5     }  
6 }  
7
```
- ProgramAp.java** (background):

```
1 public  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11 }  
12
```

The code illustrates visibility levels and overriding. In Program1, the `display()` method is `protected`. In Program2, it is overridden as `public`, which increases its visibility. In Program3, it is overridden again as `void`, which decreases its visibility. The IDE highlights the `display()` methods in different colors (blue for Program2 and red for Program3) and shows error markers for the `void` declaration in Program3.



```

class TeacherApp
{
    public void main(String[] args)
    {
        Teacher ref;
        JavaTeacher jt = new JavaTeacher();
        ref = jt;
        ref.teach();
        ref.project();
        SQLTeacher st = new SQLTeacher();
        ref = st;
        ref.teach();
        ref.project();
    }
}
  
```

2 : 4  
1 : 2  
1:M