

```

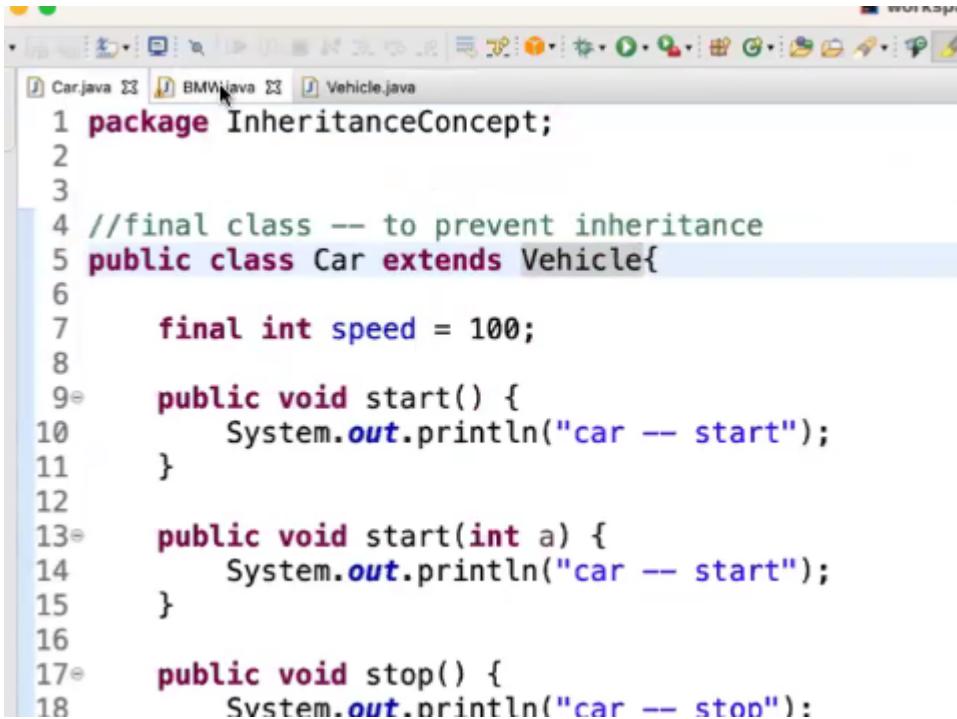
  1  package com.abc;
  2
  3  public class Car {
  4
  5      int speed = 200;
  6
  7
  8      // Method Overriding: poly(many)+Morphism(forms): RunTime(dynamic)
  9      // when we have a method in parent class and the same method in child class:
 10      // 1. with the same name
 11      // 2. with the same number of parameters
 12      // 3. with the same sequence of parameters
 13      // 4. buss logic/numbers of lines in the method -- doesn't matter
 14      // 5. with the same return type
 15
 16      //static methods can not be overridden but can be overloaded
 17
 18      @Override
  
```

Final, static, private methods cannot be overridden.

for static, private method the error is -  
*//The method billing() of type bmw1 must override or implement a supertype method*  
 for final method the error is - *Cannot override the final method from car1*

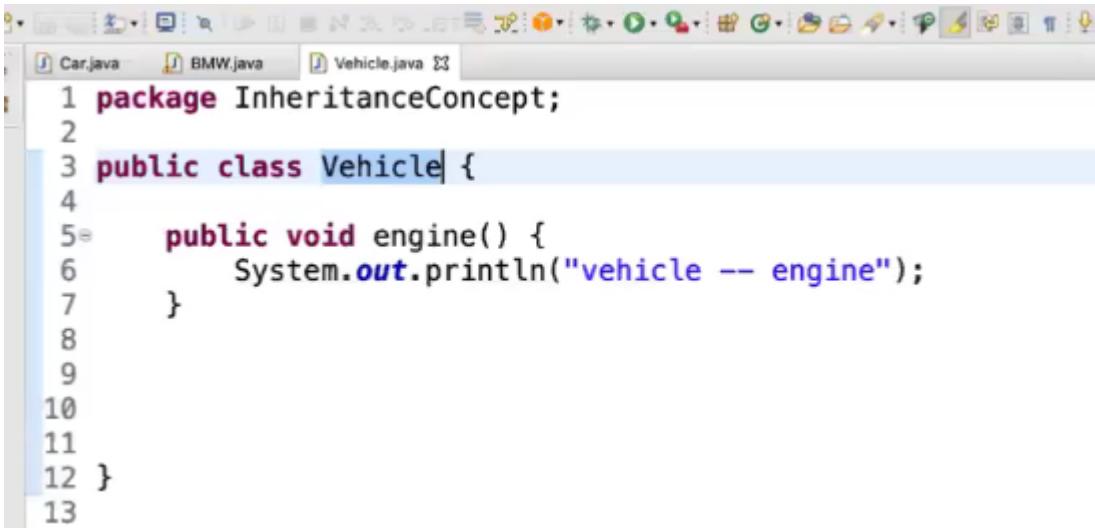


```
Car.java  BMW.java 23
6     int speed = 200;
7
8     // Method Overriding: poly(many)+Morphism(forms): RunTime(dynamic)
9     // when we have a method in parent class and the same method in child class:
10    // 1. with the same name
11    // 2. with the same number of parameters
12    // 3. with the same sequence of parameters
13    // 4. buss logic/numbers of lines in the method -- doesn't matter
14    // 5. with the same return type
15
16    //static methods can not be overridden but can be overloaded
17
18@override
19 public void start() {
20     System.out.println("BMW -- start");
21 }
22
23@Override
24 public void start(int a) {
25     System.out.println("BMW -- start");
26 }
27
28@Override
29 public void autoParking() {
30     System.out.println("BMW -- auto parking");
31 }
32
33@Override
34 public void applyBreak() {
35     System.out.println("car -- applyBreak");
36 }
37
38//method hiding
39public static void billing() {
40     System.out.println("BMW -- billing");
41 }
42
43private void getCarInfo() {
44     System.out.println("BMW -- get info");
45 }
46
47public void speedTracking() {
48     System.out.println("BMW -- speedTracking");
49 }
50
51
52 }
```



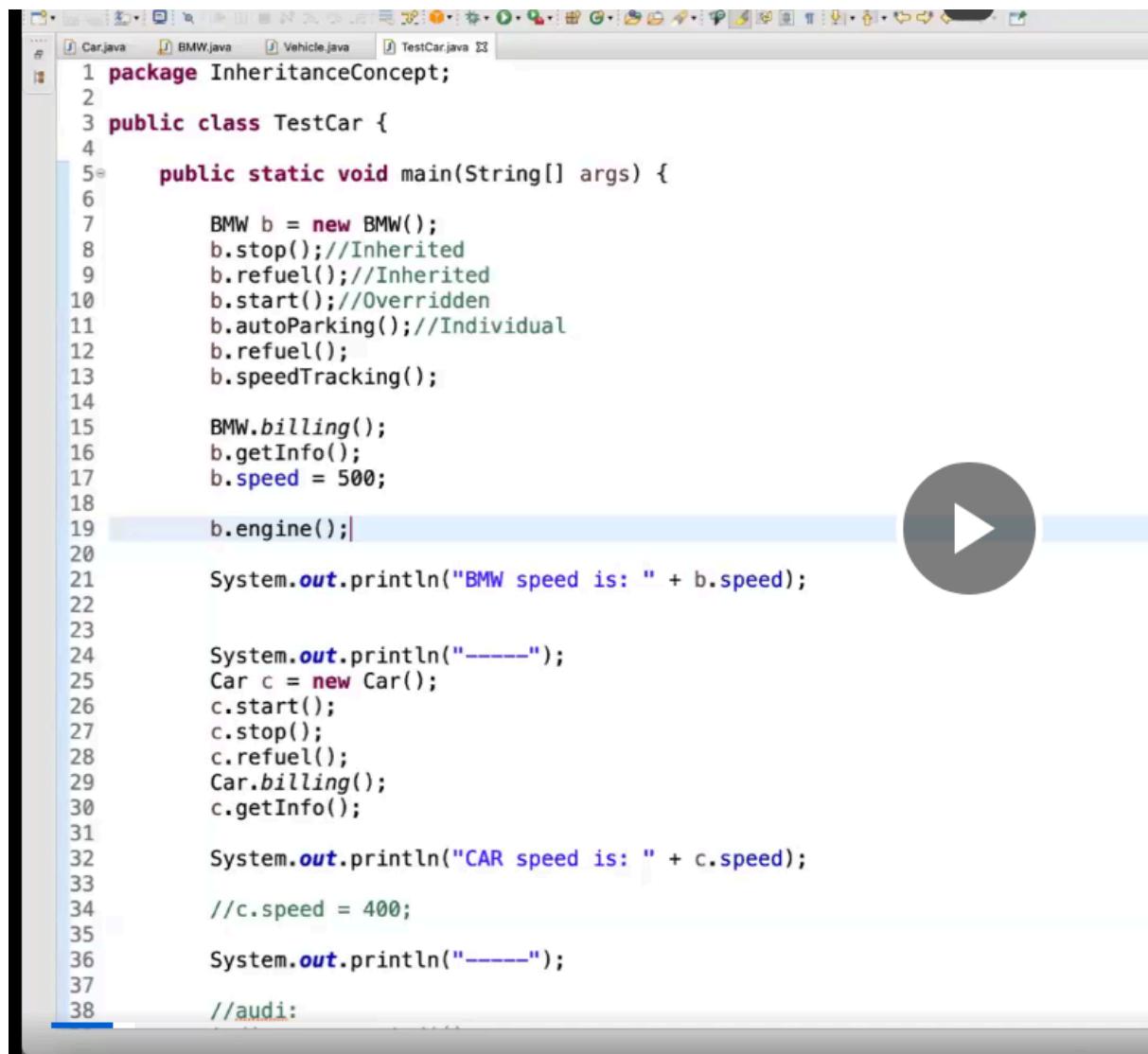
Car.java

```
1 package InheritanceConcept;
2
3
4 //final class -- to prevent inheritance
5 public class Car extends Vehicle{
6
7     final int speed = 100;
8
9     public void start() {
10         System.out.println("car -- start");
11     }
12
13     public void start(int a) {
14         System.out.println("car -- start");
15     }
16
17     public void stop() {
18         System.out.println("car -- stop");
19     }
20 }
```



Vehicle.java

```
1 package InheritanceConcept;
2
3 public class Vehicle {
4
5     public void engine() {
6         System.out.println("vehicle -- engine");
7     }
8
9
10
11
12 }
13 }
```



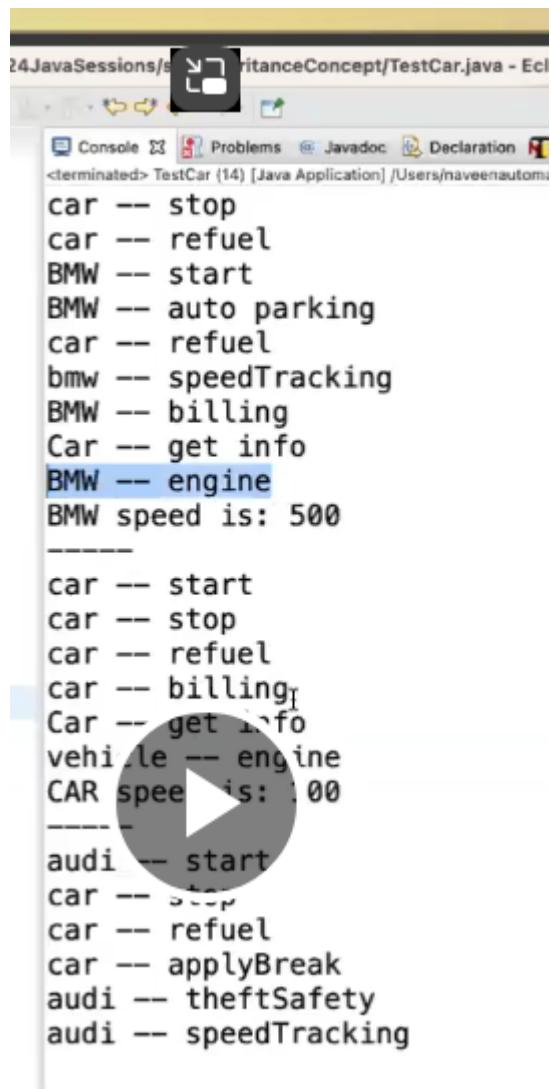
```
1 package InheritanceConcept;
2
3 public class TestCar {
4
5     public static void main(String[] args) {
6
7         BMW b = new BMW();
8         b.stop(); //Inherited
9         b.refuel(); //Inherited
10        b.start(); //Overridden
11        b.autoParking(); //Individual
12        b.refuel();
13        b.speedTracking();
14
15        BMW.billing();
16        b.getInfo();
17        b.speed = 500;
18
19        b.engine();
20
21        System.out.println("BMW speed is: " + b.speed);
22
23
24        System.out.println("----");
25        Car c = new Car();
26        c.start();
27        c.stop();
28        c.refuel();
29        Car.billing();
30        c.getInfo();
31
32        System.out.println("CAR speed is: " + c.speed);
33
34        //c.speed = 400;
35
36        System.out.println("----");
37
38        //audi:
```

```
024JavaSessions/superInheritanceConcept/TestCar.java -  
Console Problems Javadoc Declaration  
<terminated> TestCar (14) [Java Application] /Users/haveenaut  
car -- stop  
car -- refuel  
BMW -- start  
BMW -- auto parking  
car -- refuel  
bmw -- speedTracking  
BMW -- billing  
Car -- get info  
I vehicle -- engine  
BMW speed is: 500  
----  
car -- start  
car -- stop  
car -- refuel  
car -- billing  
Car -- get info  
CAR speed is: 100  
----  
audi -- start  
car -- stop  
car -- refuel  
car -- applyBreak  
audi -- theftSafety  
audi -- speedTracking
```

```
    c.stop();
    c.refuel();
    Car.billing();
    c.getInfo();
    c.engine();
}
System.out.println("
```

## Bmw override-

```
36  
37     @Override  
38     public void engine() {  
39         System.out.println("BMW -- engine");  
40     }  
41  
42     b.getinto();  
43     b.speed = 500;  
44     b.engine();  
45 }
```



The screenshot shows a terminal window within the Eclipse IDE interface. The title bar reads "24JavaSessions/... InheritanceConcept/TestCar.java - Ecl". The terminal window displays the following output:

```
car -- stop
car -- refuel
BMW -- start
BMW -- auto parking
car -- refuel
bmw -- speedTracking
BMW -- billing
Car -- get info
BMW -- engine
BMW speed is: 500
-----
car -- start
car -- stop
car -- refuel
car -- billing
Car -- get info
vehicle -- engine
CAR speed is: 100
-----
audi -- start
car -- stop
car -- refuel
car -- applyBreak
audi -- theftSafety
audi -- speedTracking
```

paste car2, vehicle2, bmw2, testfor2-

The screenshot shows a Java code editor window with the file 'vehicle2.java' open. The code defines a class 'vehicle2' with several methods: 'startengine()', 'stopengine(String name)', 'billing1()', 'getvehicledevice()', and 'getvehiclename(String name)'. The code uses System.out.println() for output. The code editor has a line number column on the left and syntax highlighting for Java keywords.

```
1 package com.day18;
2
3 public class vehicle2 {
4
5     public void startengine() {
6         System.out.println("vehicle start engine method");
7     }
8
9     public int stopengine(String name) {
10        System.out.println("vehicle start engine method");
11        return 10;
12    }
13
14     public static void billing1() {
15        System.out.println("vehicle static method");
16    }
17
18     private void getvehicledevice() {
19        System.out.println("vehicle private method.");
20    }
21
22     public final int getvehiclename(String name) {
23        System.out.println("final vehicle method");
24        return 23324;
25    }
26
27 }
28
```

```
1 package com.day18;
2
3 public class car2 extends vehicle2 {
4
5     int speed =100;
6
7     public void start() {
8         System.out.println("car -- start");
9     }
10
11    public void start(int a) {
12        System.out.println("car -- start with parameter");
13    }
14
15    public void applyBreak() {
16        System.out.println("car -- applyBreak");
17    }
18
19    //static method.
20    public static void billing() {
21        System.out.println("car -- billing");
22    }
23
24    //private method.
25    private void getCarInfo() {
26        System.out.println("car -- get info");
27    }
28
29    public final int getdetails(String name) {
30        System.out.println("car ---- get details method");
31        return 98;
32    }
33
34
35 }
36
```

```
1 package com.day18;
2
3 public class bmw2 extends car2{
4
5     @Override
6     public void start() {
7         System.out.println("bmw -- start");
8     }
9
10    @Override
11    public void start(int a) {
12        System.out.println("bmw -- start with parameter");
13    }
14
15    public void autoparking() {
16        System.out.println("bmw auto parking");
17    }
18
19    @Override
20    public void applyBreak() {
21        System.out.println("bmw -- applyBreak");
22    }
23
24    public void getspeedfrombmw() {
25        System.out.println("bmw only method");
26    }
27
28    public int getnamefrombmw(String name) {
29        System.out.println("bmw only method");
30        return 878;
31    }
32
33    public static int getcarlocation(String name) {
34        System.out.println("bmw static method car location");
35        return 78;
36    }
37
38 }
39
```

```

testfor2.java
1 package com.day18;
2
3 public class testfor2 {
4
5     public static void main(String[] args) {
6
7         bmw2 b1=new bmw2();
8         //b1 can access all public, static, final of itself, parent and grandparent.
9
10        car2 c1=new car2();
11        //c1 can access all public, static, final of itself, parent and grandparent.
12        c1.
13
14        vehicle2 v1=new vehicle2();
15
16        //v1 can access all public, static, final of itself, parent and grandparent.
17
18        System.out.println("bmw can access its own static method");
19        int carlocation=bmw2.getcarlocation("tiger");
20        System.out.println(carlocation);
21
22        System.out.println("bmw can access its parent static method");
23        bmw2.billing();
24
25        System.out.println("bmw can access its grand parent static method");
26        bmw2.billing1();
27
28        System.out.println("bmw can access its grand parent static method");
29        bmw2.billing1();
30
31        //similarly for the car
32        System.out.println("car can access its own static method");
33        car2.billing();
34
35        System.out.println("car can access its parent static method");
36        car2.billing1();
37
38    }
39
40 }

41
42 //bmw can access its own static method
43 //bmw static method car location
44 //78
45 //bmw can access its parent static method
46 //car -- billing
47 //bmw can access its grand parent static method
48 //vehicle static method
49 //car can access its own static method
50 //car -- billing

```

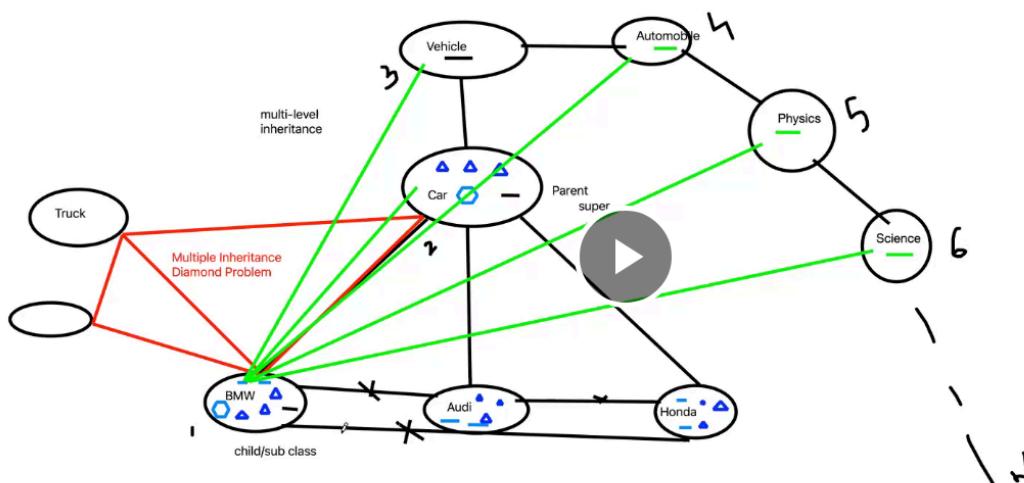
Parents cannot take from child-  
you wont get option only after dot.

```

49
50     Vehicle vh = new Vehicle();
51     vh.engine();|
52

```

No limit on number of levels-



```

54
55     //child class object can be referred by parent class reference variable: Top Casting/Up Casting
56     Car c1 = new BMW();|
57     c1.start();|

```

BMW -- start

Bmw start method is called not car class.

Car cannot access bmw individual method like auto parking.

```

59
60     c1.auto| |
61

```

This will work-

```

58         c1.stop();
59         c1.refuel();
60
parent -- start
car -- stop
car -- refuel I

```

//child class object can be referred by parent class reference variable: Top Casting/Up Casting  
//can access all the overridden and inherited methods only  
//but can not access individual methods -- ref type check will be failed  
Car c1 = new BMW();

## Down casting-

Not allowed in java. Error.

```

63     //Down casting:
64     //parent class object can be referred by child class reference variable: Down Casting
65
66     BMW b1 = new Car();
67

```

Every car is not bmw so error.

Type mismatch: cannot convert from car3 to bmw3

```

65
66     BMW b1 = (BMW) new Car();
67

```

Cast it. Now error comes at run time. Class cast exception. run the code for it to come.

```

Exception in thread "main" java.lang.ClassCastException: class InheritanceConcept.Car
at InheritanceConcept.TestCar.main(TestCar.java:66)

```

## Is a relationship-

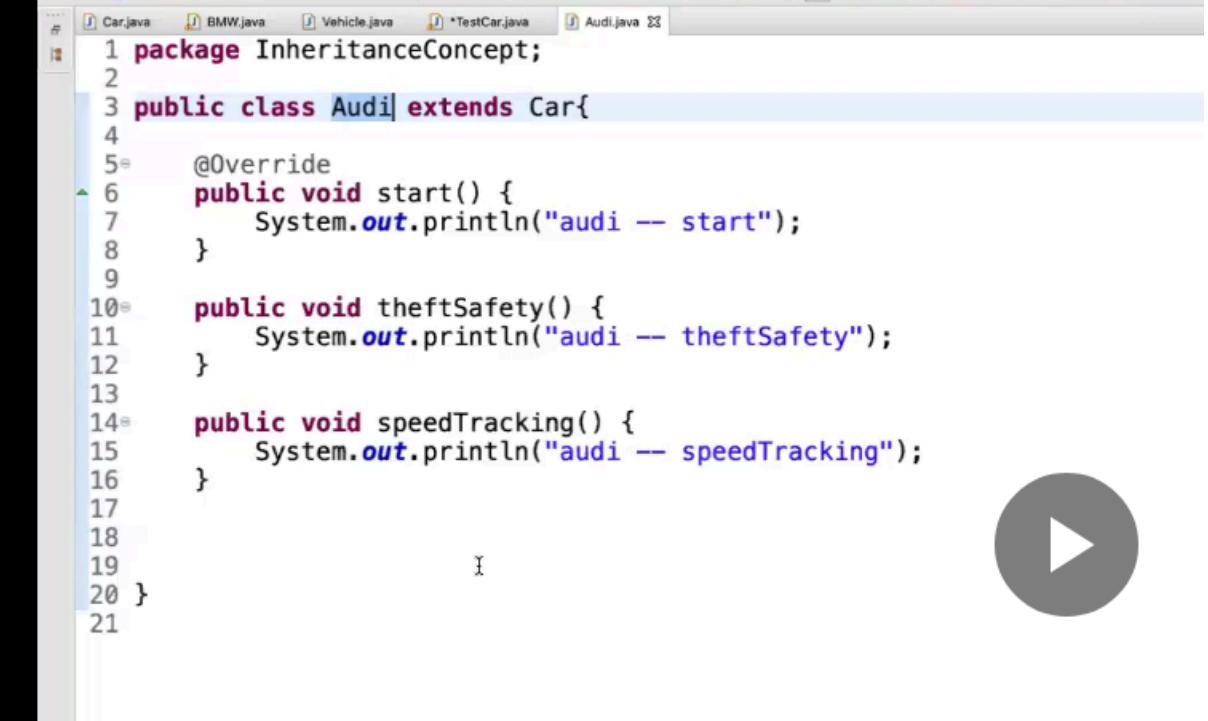
```

54
55     //child class object can be referred by parent class reference variable: Top Casting/Up Casting
56     //can access all the overridden and inherited methods only
57     //but can not access individual methods -- ref type check will be failed
58     Car c1 = new BMW(); //IS_A relationship
59

```

In top casting and in inheritance.

## Audi car-



The screenshot shows a Java code editor with several tabs at the top: Car.java, BMW.java, Vehicle.java, \*TestCar.java, and Audi.java. The Audi.java tab is active. The code is as follows:

```
1 package InheritanceConcept;
2
3 public class Audi extends Car{
4
5     @Override
6     public void start() {
7         System.out.println("audi -- start");
8     }
9
10    public void theftSafety() {
11        System.out.println("audi -- theftSafety");
12    }
13
14    public void speedTracking() {
15        System.out.println("audi -- speedTracking");
16    }
17
18
19 }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63     Car c2 = new Audi();
64     c2.start();
65     c2.stop();
66     c2.refuel();
67 }
```

A large circular play button icon is positioned to the right of the code editor.

Note-

Parents can access the methods which are overridden from them because they also have the same methods..

```

    workspace1 - May 2024 Java Sessions/src/InheritanceConcept/TestCar.java - Eclipse IDE
    67
    68
    69
    70     //System.out.println("----");
    71     //child class object can be referred by grand parent class reference variable: Top Casting/Up Casting
    72     Vehicle v2 = new BMW(); //IS_A relationship
    73     v2.engine();
    74
  
```

Diagram illustrating Java inheritance and casting:

- Vehicle** class (superclass)
- BMW** class (child class)
- Car** class (another child class)
- Vehicle** reference variable (`v1`)
- BMW** reference variable (`v2`)
- Car** reference variable (`v3`)

Annotations in the diagram:

- Vehicle** reference variable (`v1`):  
→ Can refer to **BMW** objects because **BMW** is a child of **Vehicle**. This is called **Top Casting/Up Casting**.
- Vehicle** reference variable (`v1`):  
→ Can refer to **Car** objects because **Car** is also a child of **Vehicle**. This is called **Top Casting/Up Casting**.
- BMW** reference variable (`v2`):  
→ Can be referred by **Vehicle** reference variable (`v1`). This is called **IS\_A relationship**.
- Car** reference variable (`v3`):  
→ Can be referred by **Vehicle** reference variable (`v1`). This is called **IS\_A relationship**.

## Downcast to grandparent-

```

    79
    80         BMW b3 = (BMW) new Vehicle();
    81
    82
  
```

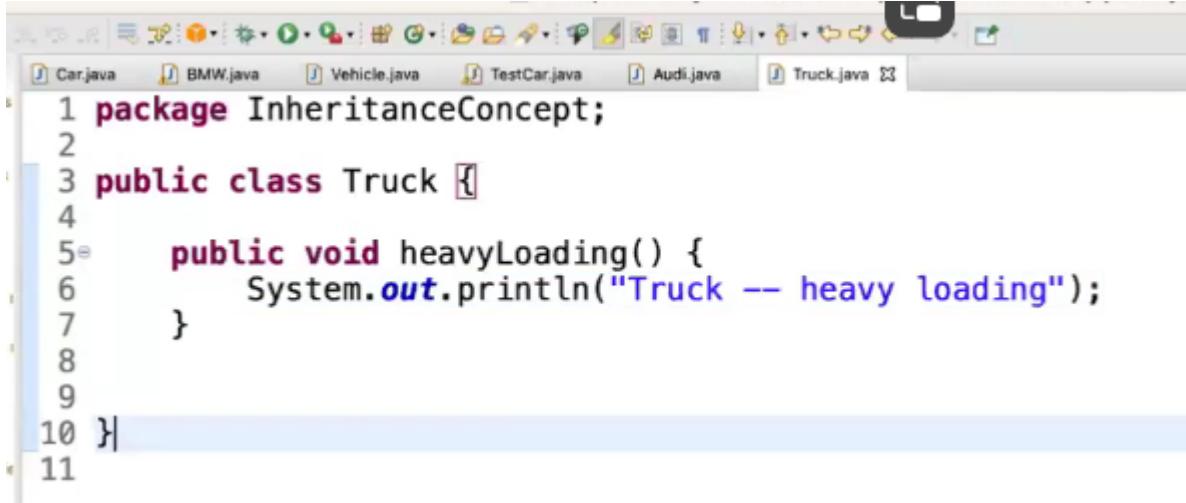
Class cast exception. we get this after running the code. //after running we get:

```

    //Exception in thread "main" java.lang.ClassCastException:
    //class com.day18.vehicle5 cannot be cast to class
    com.day18.bmw5
    // (com.day18.vehicle5 and com.day18.bmw5 are in unnamed
    module of loader 'app')
  
```

```
//at com.day18.testfor5.main(testfor5.java:11)
```

## Truck class-

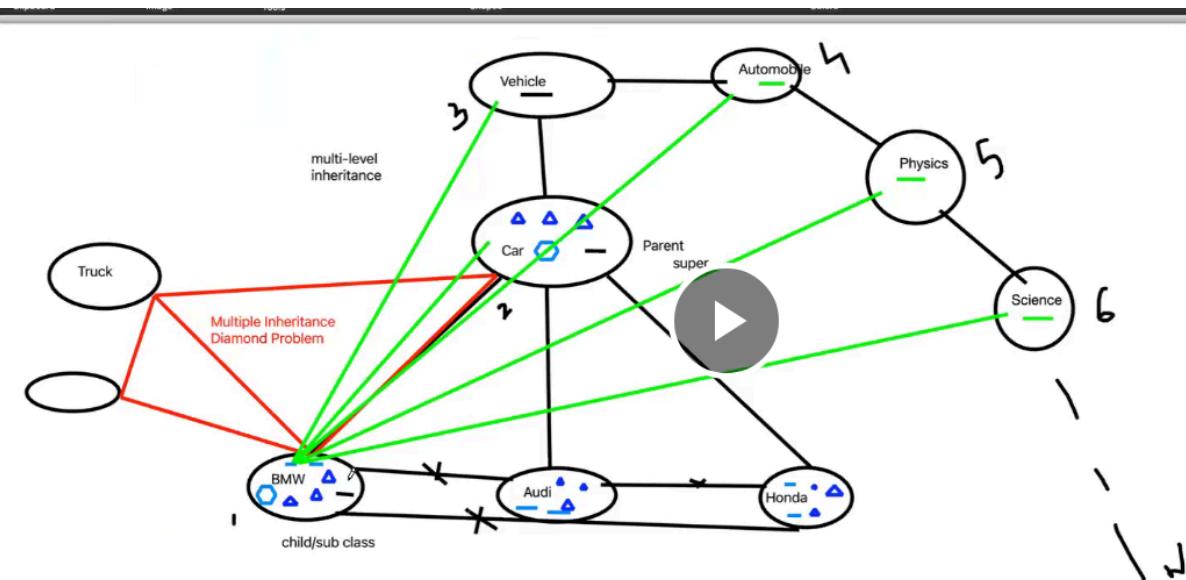


```

1 package InheritanceConcept;
2
3 public class Truck {
4
5     public void heavyLoading() {
6         System.out.println("Truck -- heavy loading");
7     }
8
9
10}
11

```

## Multiple inheritance not allowed-



Not allowed error. //Syntax error on token(s), misplaced construct(s)

```

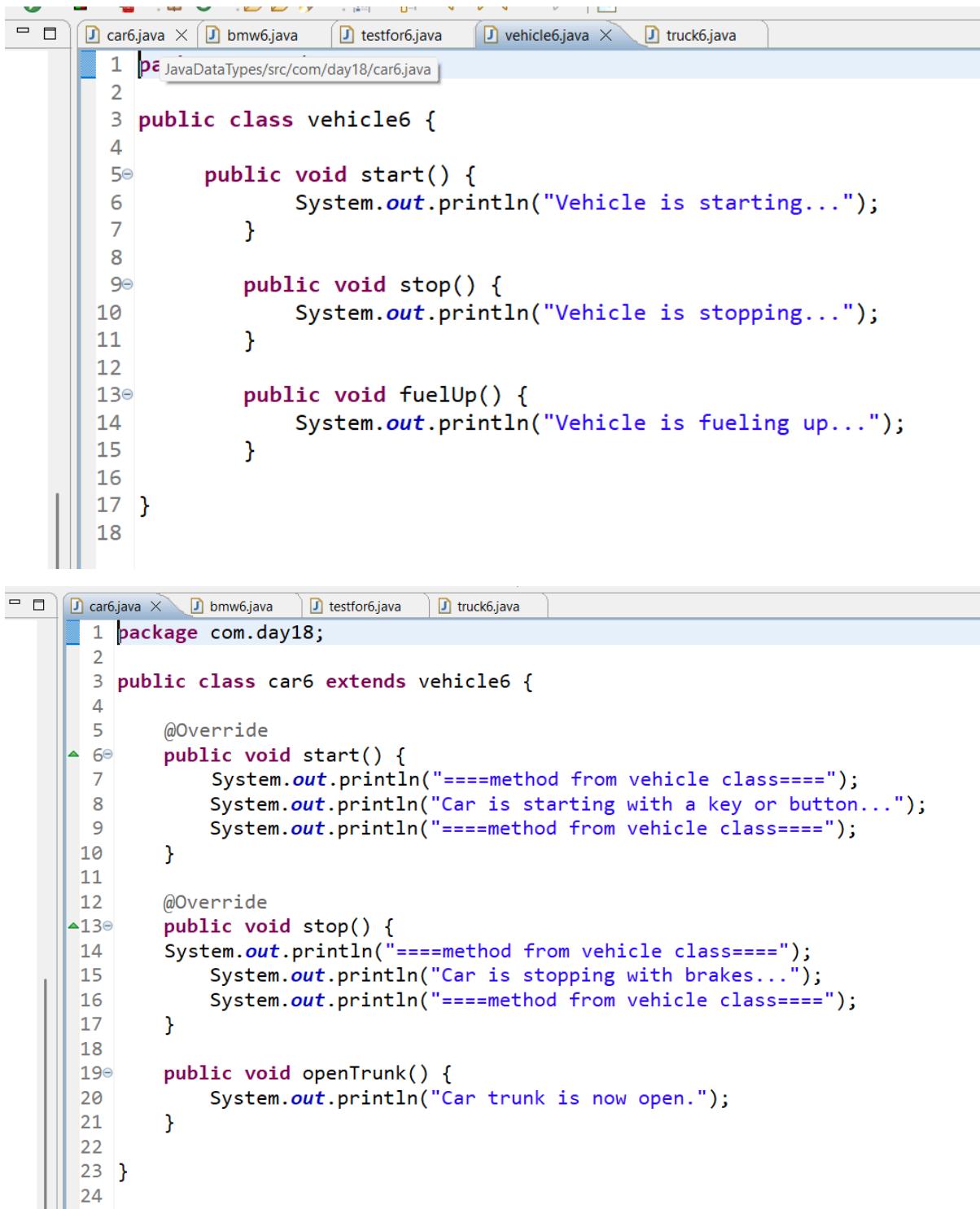
2
3 public class BMW extends Car, Truck {
4

```

```
58
59  public void bmwLoading() {
60      Truck tr = new Truck(); //BMW class is having Truck class object and able to access its methods:Composition
61      tr.heavyLoading();
62  }
19          b.engine();
20          b.bmwLoading();

BMW -- engine
Truck + heavy loading
BMW speed is 500
```

paste car6, vehicle6, bmw6, truck6, testfor6-



```

1 package JavaDataTypes/src/com/day18/car6.java
2
3 public class vehicle6 {
4
5     public void start() {
6         System.out.println("Vehicle is starting...");
7     }
8
9     public void stop() {
10        System.out.println("Vehicle is stopping...");
11    }
12
13    public void fuelUp() {
14        System.out.println("Vehicle is fueling up...");
15    }
16
17 }
18

```

```

1 package com.day18;
2
3 public class car6 extends vehicle6 {
4
5     @Override
6     public void start() {
7         System.out.println("====method from vehicle class====");
8         System.out.println("Car is starting with a key or button...");
9         System.out.println("====method from vehicle class====");
10    }
11
12    @Override
13    public void stop() {
14        System.out.println("====method from vehicle class====");
15        System.out.println("Car is stopping with brakes...");
16        System.out.println("====method from vehicle class====");
17    }
18
19    public void openTrunk() {
20        System.out.println("Car trunk is now open.");
21    }
22
23 }
24

```

```
bmw6.java testfor6.java truck6.java
1 package com.day18;
2
3 //Syntax error on token(s), misplaced construct(s)
4 //public class bmw6 extends car6, truck6{
5
6 public class bmw6 extends car6{
7
8     @Override
9     public void start() {
10         System.out.println("====method from car class====");
11         System.out.println("BMW is starting with push-button and advanced system... ");
12         System.out.println("====method from car class====");
13     }
14
15     @Override
16     public void fuelUp() {
17         System.out.println("====method from vehicle class====");
18         System.out.println("BMW requires premium fuel.");
19         System.out.println("====method from vehicle class====");
20     }
21
22     public void activateSportMode() {
23         System.out.println("BMW is now in Sport Mode!");
24     }
25
26     //how to call sibling class methods and variables.
27     public void callsibling() {
28         truck6 t1=new truck6();
29         int v1= t1.truck;
30         System.out.println(v1);
31         t1.loadCargo();
32         t1.loadCargo("karan");
33     }
34
35 }
36
```

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

- truck6.java** (Top Tab):
 

```

1 package com.day18;
2
3 public class truck6 {
4
5     int truck=100;
6
7     public void loadCargo() {
8         System.out.println("Truck is loading cargo...");
9     }
10
11    public void loadCargo(String name) {
12        System.out.println("Truck is loading cargo..." + " " + name);
13    }
14
15 }
16 
```
- testfor6.java** (Bottom Tab):
 

```

1 package com.day18;
2
3 public class testfor6 {
4
5     public static void main(String[] args) {
6
7         bmw6 b1=new bmw6();
8         b1.callsibling();
9     }
10
11 }
12
13 //100
14 //Truck is loading cargo...
15 //Truck is loading cargo... karan
16 
```

## Add variable-

```

2
3 public class Vehicle {
4
5     int price = 50;
6
7     System.out.println("BMW Speed is: " + b.speed);
8     System.out.println("BMW price is : " + b.price);
9 
```

```
BMW price is : 50
---
```

Add same variable in child-

```
3
4 //final class -- to prevent inheritance
5 public class Car extends Vehicle{
6
7     final int speed = 100; I
8     int price = 70;
9
12         System.out.println("BMW speed is: " + b.speed);
13         System.out.println("BMW price is : " + b.price); |
```



```
BMW price is : 70
---
```

Recent most value is captured. Also the immediate parent is given preference.

Bmw uses same variable and increased price-

```
Car.java  BMW.java  Vehicle.java  TestCar.java  Audi.java
1 package InheritanceConcept;
2
3 public class BMW extends Car {
4
5     //bmw class var
6     int speed = 200;
7     int price = 500;
8
22         System.out.println("BMW speed is: " + b.speed);
23         System.out.println("BMW price is : " + b.price); |
```

```
BMW speed is: 200
BMW price is : 500
---
```

Bmw value is taken.

Car price-

```

35     System.out.println("CAR speed is: " + c.speed);
36     System.out.println("CAR price is : " + c.price);
37

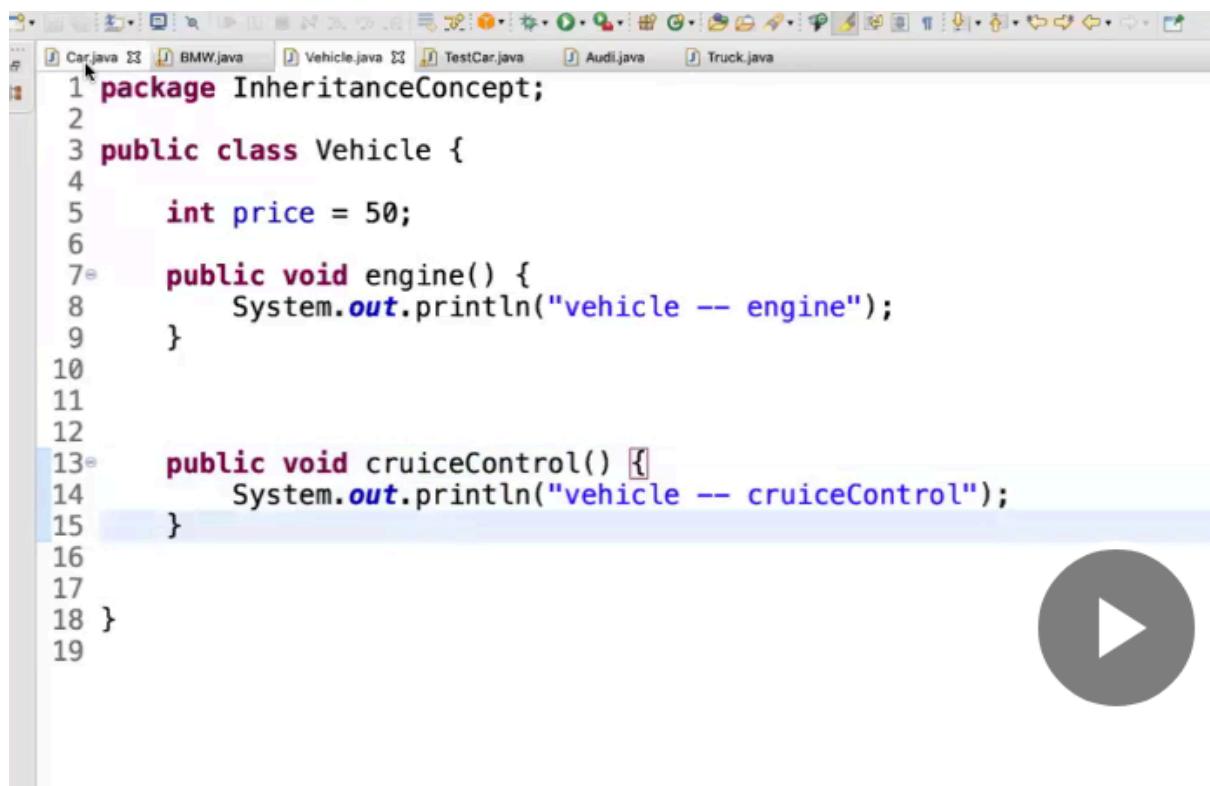
```

70.

Note-

First own class is checked, then one level up, if not there then one more level up.

One more method-



```

1 package InheritanceConcept;
2
3 public class Vehicle {
4
5     int price = 50;
6
7     public void engine() {
8         System.out.println("vehicle -- engine");
9     }
10
11
12
13     public void cruiseControl() {
14         System.out.println("vehicle -- cruiseControl");
15     }
16
17
18 }
19

```

```

10
11
12
13
14
15
16
17     b.speed = 500;
18
19     b.engine();
20     b.bmwLoading();
21     b.cruiseControl();
22

```

Cruise control from vehicle printed.

Car overrides-

```

45
46  @Override
47  public void cruiceControl() {
48      System.out.println("car -- cruiceControl");
49  }
50

51      b.bmwLoading();
52      b.cruiceControl();
```

Car cruise control is called.

Bmw cruise control-

```

66  @Override
67  public void cruiceControl() {
68      System.out.println("BMW -- cruiceControl");
69  }
70

71      b.speed = 500;
72
73      b.engine();
74      b.bmwLoading();
75      b.cruiceControl();
```

Bmw cruise control.

```

34      c.engine();
35      c.cruiceControl();
```

Car cruise control.

method chaining–

```
28     }
29
30+ public void autoParking() {
31     System.out.println("BMW -- auto parking");
32     stop(); //Inheritance
33 }
34
35+ @Override
36
37
38 public void autoParking() {
39     System.out.println("BMW -- auto parking");
40     applyBreak(); //method chaining
41     stop(); //Inheritance
42 }
```

paste vehicle14, car14, bmw14, testfor14 -

The screenshot shows a Java development environment with two code editors side-by-side.

**Top Editor (vehicle14.java):**

```
1 package com.day18;
2
3 public class vehicle14 {
4
5     String type = "Generic Vehicle";
6
7     //create one simple method here.
8     public void startengine() {
9         System.out.println("vehicle start engine method");
10    }
11
12     //call one method with different name and call one method from bmw14 and one method from car14.
13     public void karanmethod1() {
14         bmw14 b1=new bmw14();
15         b1.method5();
16         car14 c1=new car14();
17         c1.karanmethod();
18     }
19
20 }
21
```

**Bottom Editor (car14.java):**

```
1 package com.day18;
2
3 public class car14 extends vehicle14 {
4
5     String type = "Car CLASs";
6
7     //override the same method from vehicle11.
8     @Override
9     public void startengine() {
10        System.out.println("car start engine method");
11    }
12
13     //create one method and call the bmw14 method here.
14     public void karanmethod() {
15         bmw14 b1=new bmw14();
16         b1.method5();
17     }
18
19 }
20
```

```
1 package com.day18;
2
3 public class bmw14 extends car14{
4
5     //we can chain any method with anything.
6     //overridden methods can also be chained.
7
8     String type = "bmw CLASs";
9
10    //override method from car12.
11    @Override
12    public void startengine() {
13        System.out.println("bmw start engine method");
14        method1();
15        method2();
16        method3();
17        method4();
18    }
19
20    //create another four methods.
21
22    public void method1() {
23        System.out.println("method1");
24    }
25    public void method2() {
26        System.out.println("method2");
27    }
28    public static void method3() {
29        System.out.println("method3");
30    }
31    public final void method4() {
32        System.out.println("method4");
33    }
34
35    //create one method which calls all the above methods inside it.
36    public void method5() {
37        method1();
38        method2();
39        method3();
40        method4();
41        startengine();
42    }
43
44 }
45 }
```

The screenshot shows a Java code editor window with the file 'testfor14.java' open. The code is as follows:

```
1 package com.day18;
2
3 public class testfor14 {
4
5     public static void main(String[] args) {
6
7         //create object of bmw14.
8         bmw14 b1=new bmw14();
9         b1.method5();
10        b1.startengine();
11
12        b1.karanmethod();
13        b1.karanmethod1();
14
15    }
16
17 }
18
19 //method1
20 //method2
21 //method3
22 //method4
23 //bmw start engine method
24 //method1
25 //method2
26 //method3
27 //method4
28 //bmw start engine method
29 //method1
30 //method2
```

```
29 //method1
30 //method2
31 //method3
32 //method4
33 //method1
34 //method2
35 //method3
36 //method4
37 //bmw start engine method
38 //method1
39 //method2
40 //method3
41 //method4
42 //method1
43 //method2
44 //method3
45 //method4
46 //bmw start engine method
47 //method1
48 //method2
49 //method3
50 //method4
51 //method1
52 //method2
53 //method3
54 //method4
55 //bmw start engine method
```

```
54 //method4  
55 //bmw start engine method  
56 //method1  
57 //method2  
58 //method3  
59 //method4  
60  
61  
62
```

Final and static methods can be inherited but not overridden.

paste car15, vehicle15.

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

- vehicle15.java** (Top Window):
 

```

1 package com.day18;
2
3 public class vehicle15 {
4
5     final String type = "Generic Vehicle";
6
7     static int i=100;
8
9     //create two final methods one with return and other no return.
10    public final int getvehiclename(String name) {
11        System.out.println("final vehicle method");
12        return 23324;
13    }
14
15    public final void startengine() {
16        System.out.println("vehicle start engine method");
17    }
18
19    //create two static methods one with return and other no return.
20    public static int getvehiclename1(String name) {
21        System.out.println("final vehicle method");
22        return 23324;
23    }
24
25    public static void startengine1() {
26        System.out.println("vehicle start engine method");
27    }
28}
29
      
```
- car15.java** (Bottom Window):
 

```

1 package com.day18;
2
3 public class car15 extends vehicle15 {
4
5     final String type = "Generic Vehicle";
6
7     static int i=100;
8
9     //override the final and static variables from vehicle15.
10    //Cannot override the final method from vehicle15
11    // @Override
12    // public final void startengine() {
13    //     System.out.println("car start engine method");
14    // }
15
16    //The method startengine1() of type car15 must override or implement a supertype method
17    @Override
18    public static void startengine1() {
19        System.out.println("car start engine method");
20    }
21
22}
23
      
```

paste vehicle16, car16.

The screenshot shows an IDE interface with two tabs at the top: "car16.java" and "vehicle16.java". The "vehicle16.java" tab is active, displaying the following code:

```

1 package com.day18;
2
3 public class vehicle16 {
4
5     final String type = "Generic Vehicle";
6
7     static int i=100;
8
9     //create two final methods one with return and other no return.
10    public final int getvehiclename(String name) {
11        System.out.println("final vehicle method");
12        return 23324;
13    }
14
15    public final void startengine() {
16        System.out.println("vehicle start engine method");
17    }
18
19     //create two static methods one with return and other no return.
20    public static int getvehiclename1(String name) {
21        System.out.println("final vehicle method");
22        return 23324;
23    }
24
25    public static void startengine1() {
26        System.out.println("vehicle start engine method");
27    }
28 }
29

```

The "car16.java" tab is also visible, showing the following code:

```

1 package com.day18;
2
3 public class car16 extends vehicle15 {
4
5     final String type = "Generic Vehicle";
6
7     static int i=100;
8
9     //even without overload annotation the same error for final methods.
10
11    //Cannot override the final method from vehicle15
12    public final void startengine() {
13        System.out.println("car start engine method");
14    }
15
16    //method hiding.
17    public static void startengine1() {
18        System.out.println("car start engine method");
19    }
20
21 }
22

```

paste vehicle17, car17, testfor17-

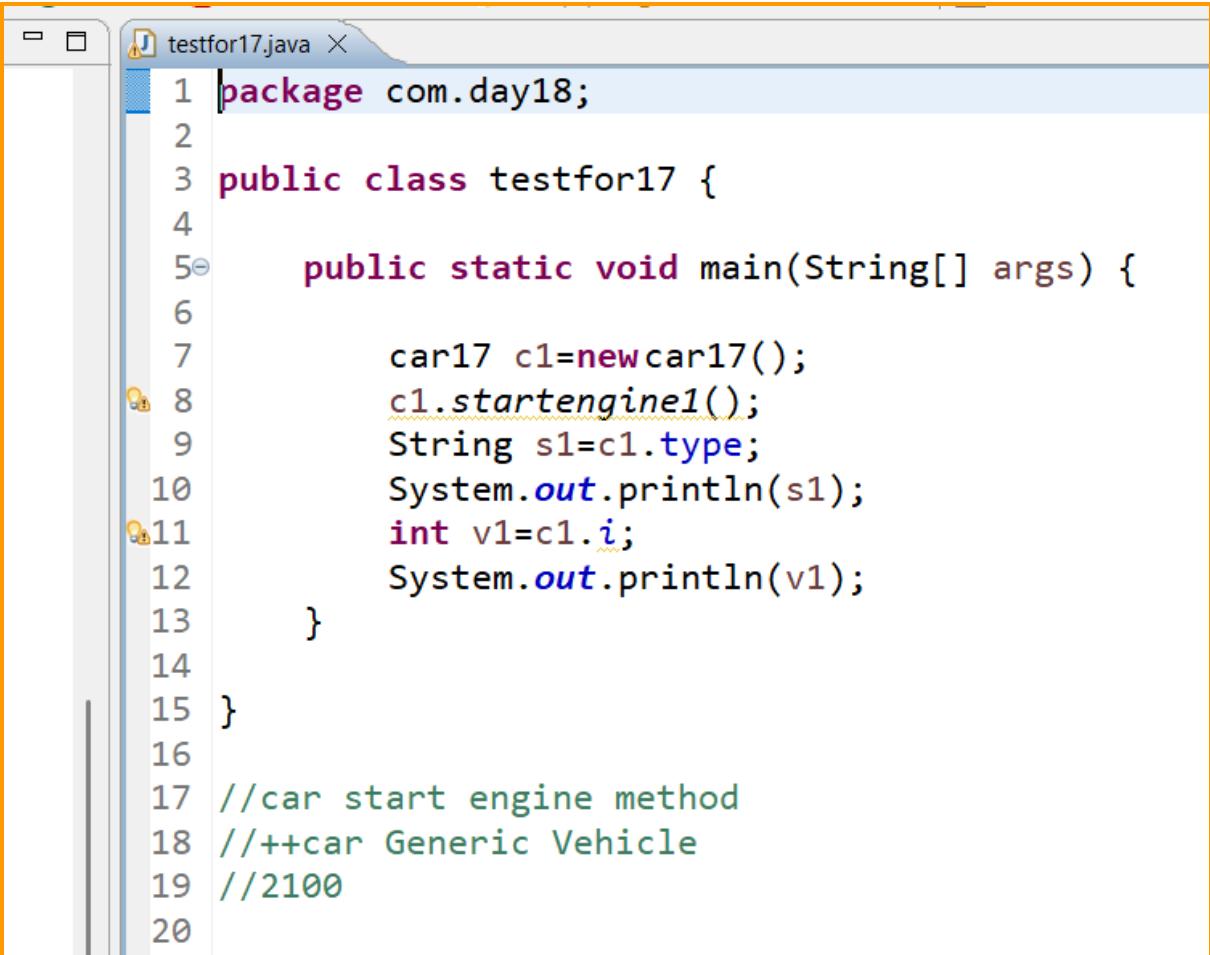
The screenshot shows a Java development environment with two code editors side-by-side.

**Top Editor (vehicle17.java):**

```
1 package com.day18;
2
3 public class vehicle17 {
4
5     final String type = "Generic Vehicle";
6
7     static int i=100;
8
9     //create two final methods one with return and other no return.
10    public final int getvehiclename(String name) {
11        System.out.println("final vehicle method");
12        return 23324;
13    }
14
15    public final void startengine() {
16        System.out.println("vehicle start engine method");
17    }
18
19     //create two static methods one with return and other no return.
20    public static int getvehiclename1(String name) {
21        System.out.println("final vehicle method");
22        return 23324;
23    }
24
25    public static void startengine1() {
26        System.out.println("vehicle start engine method");
27    }
28}
29
```

**Bottom Editor (car17.java):**

```
1 package com.day18;
2
3 public class car17 extends vehicle15 {
4
5     final String type = "++car Generic Vehicle";
6
7     static int i=2100;
8
9     //method hiding.
10    public static void startengine1() {
11        System.out.println("car start engine method");
12    }
13
14}
15
```



```

1 package com.day18;
2
3 public class testfor17 {
4
5     public static void main(String[] args) {
6
7         car17 c1=newcar17();
8         c1.startengine1();
9         String s1=c1.type;
10        System.out.println(s1);
11        int v1=c1.i;
12        System.out.println(v1);
13    }
14
15 }
16
17 //car start engine method
18 //++car Generic Vehicle
19 //2100
20

```

## Calling final methods-

```

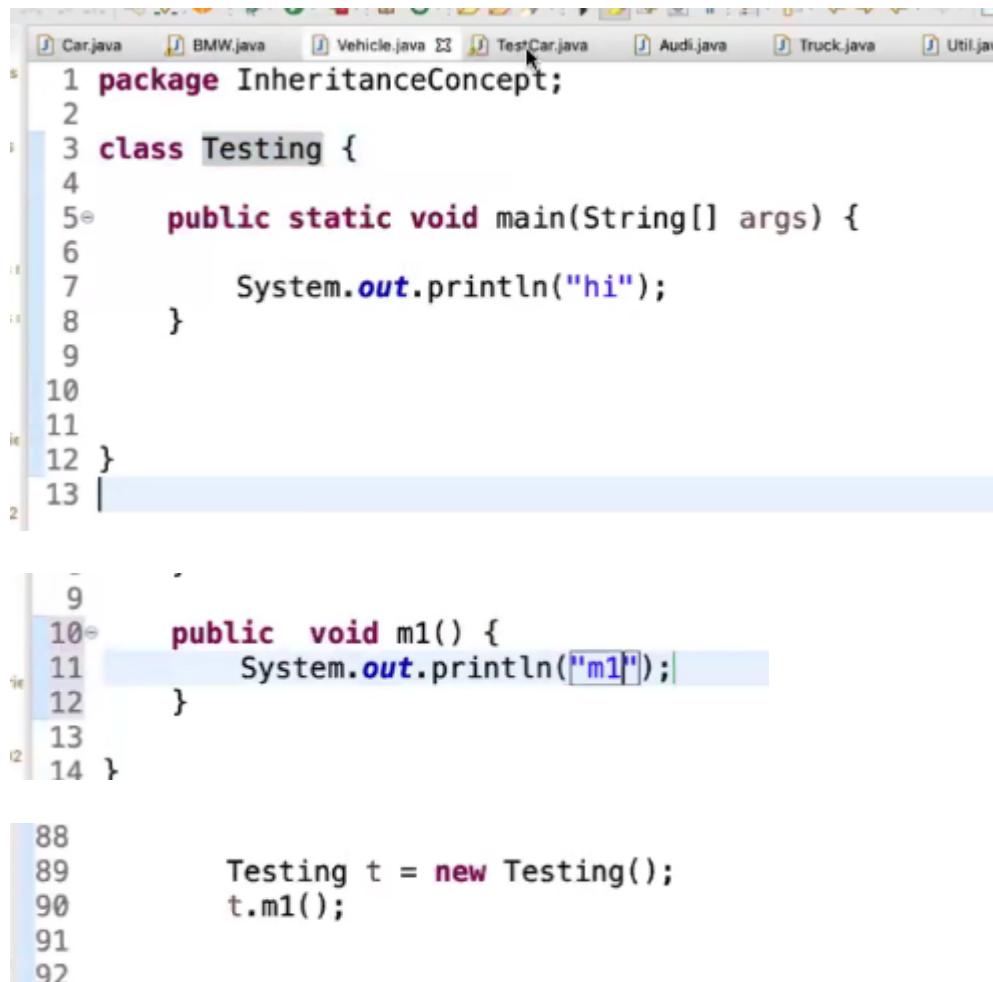
62
63     Audi au = new Audi();
64     au.speedTracking();
65

```

Audi speed tracking.

Using composition we can access child class method in parent and grand parents methods. We create objects.

## Default class-



```

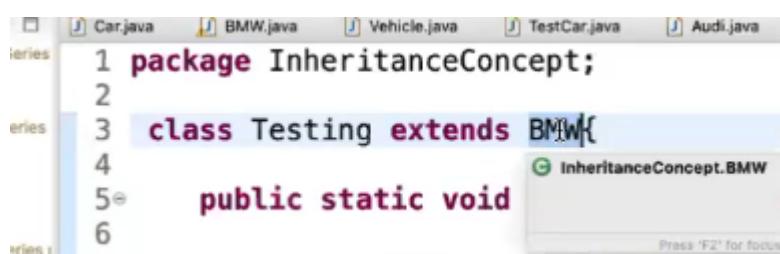
1 package InheritanceConcept;
2
3 class Testing {
4
5     public static void main(String[] args) {
6
7         System.out.println("hi");
8     }
9
10
11 }
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92

```

Works fine.

M1 is the output.

Default class can extend another class –



```

1 package InheritanceConcept;
2
3 class Testing extends BMW {
4
5     public static void
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92

```

paste default2, bmw14-

```
1 package com.day18;
2
3 //default class can also extend another class.
4 class default2 extends bmw14 {
5
6     public static void main(String[] args) {
7
8         default2 d1=new default2();
9         d1.method5();
10        d1.startengine();
11        String name=d1.type;
12        System.out.println(name);
13        d1.karanmethod();
14        d1.karanmethod1();
15        d1.method1();
16        d1.method2();
17        d1.method4();
18        d1.startengine();
19        d1.method3(); //The static method method3() from the type bmw14 should be
20        //accessed in a static way
21
22        default2.method3(); //default can access static method by the class name.
23        bmw14.method3(); //bmw can call its static method.
24
25
26    }
27
28 }
```

```
27
28 }
29
30
31 // method1
32 // method2
33 // method3
34 // method4
35 // bmw start engine method
36 // method1
37 // method2
38 // method3
39 // method4
40 // bmw start engine method
41 // method1
42 // method2
43 // method3
44 // method4
45 // bmw CLASs
46 // method1
47 // method2
48 // method3
49 // method4
50 // bmw start engine method
51 // method1
52 // method2
53 // method3
54 // method4
```

```
53 // method3
54 // method4
55 // method1
56 // method2
57 // method3
58 // method4
59 // bmw start engine method
60 // method1
61 // method2
62 // method3
63 // method4
64 // method1
65 // method2
66 // method3
67 // method4
68 // bmw start engine method
69 // method1
70 // method2
71 // method3
72 // method4
73 // method1
74 // method2
75 // method4
76 // bmw start engine method
77 // method1
```

```
76 // bmw start engine method
77 // method1
78 // method2
79 // method3
80 // method4
81 // method3
82 // method3
83 // method3
84
85
86
87
```

```
1 package com.day18;
2
3 public class bmw14 extends car14{
4
5     //we can chain any method with anything.
6     //overridden methods can also be chained.
7
8     String type = "bmw CLASs";
9
10    //override method from car12.
11    @Override
12    public void startengine() {
13        System.out.println("bmw start engine method");
14        method1();
15        method2();
16        method3();
17        method4();
18    }
19
20    //create another four methods.
21
22    public void method1() {
23        System.out.println("method1");
24    }
25    public void method2() {
26        System.out.println("method2");
27    }
28    System.out.println("method2");
29    public static void method3() {
30        System.out.println("method3");
31    }
32    public final void method4() {
33        System.out.println("method4");
34    }
35
36    //create one method which calls all the above methods inside it.
37    public void method5() {
38        method1();
39        method2();
40        method3();
41        method4();
42        startengine();
43    }
44
45 }
```

## Note-

```
  Eclipse   File   Edit   Source   Refactor   Navigate   Search   Project   Scout   Run   Window   Help
  workspace1 - May2024JavaSessions/src/Inheritance
  Car.java  BMW.java  Vehicle.java  TestCar.java  Audi.java  Truck.java  Util.java  ElementUtil.java  Testing.java

17      System.out.println("car -- start");
18  }
19
20  public void stop() {
21      System.out.println("car -- stop");
22  }
23
24 // final methods can not be overridden -- prevent method overriding
25  public final void refuel() {
26      System.out.println("car -- refuel");
27  }
28
29  public void applyBreak() {
30      System.out.println("car -- applyBreak");
31  }
32
33 // static methods can not be overridden
34  public static void billing() {
35      System.out.println("car -- billing");
36  }
37
38 // private methods can not be overridden
39  private void getCarInfo() {
40      System.out.println("Car -- get info");
41  }
42
43  public void getInfo() {
44      getCarInfo();
45  }
46
47  @Override
48  public void cruiseControl() {
49      System.out.println("car -- cruiseControl");
50  }
51
52 }
53
54 
```

