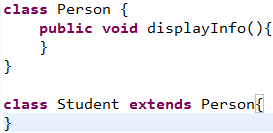
# Why can’t child class specify more restricted access Specifiers when overriding?

**Let’s try to understand through Example**:   
🡪 Suppose we have following code:  


Now inherited method displayInfo() in Student may be used outside of Student class at thousands of places and we can use it as it’s public. But In future, you decide to override this method in Student class with required implementation declaring it to be more restricted. Now this method can’t be accessed outside of Student class. And the error will occur at those thousands of places.   
🡪 **Jatin**:   
Suppose you override in Student class with more restricted specifier.  
When you compile 🡪 person.displayInf(): at compile time, compiler finds person of Person type and checks its visibility in Person which is public ok.

Now, you override displayInfo() in Student class with private modifier.   
person.displayInfo(): at Compile time, compiler checks the visibility, which is public in Person class as compiler identifies the type by reference type.   
person.displayInfo(): at runtime, JVM checks the runtime object which is Student type and checks the visibility which is private so at runtime problem may occur so to avoid this problem in future, visibility is checked by compiler at compile time.

# Which of the followings are valid or which are invalid? Rule: If child class throws checked exception, then parent class must throw the same or its parent class checked exception. But there are no restrictions on Un-checked exception. NOTE: Rule is from child to parent 🡪 if child then parent but not the other way around meaning not from parent to child like if parent throws this checked exception then child should also throw so so and so checked exception. ☺

invalid

invalid

invalid

valid

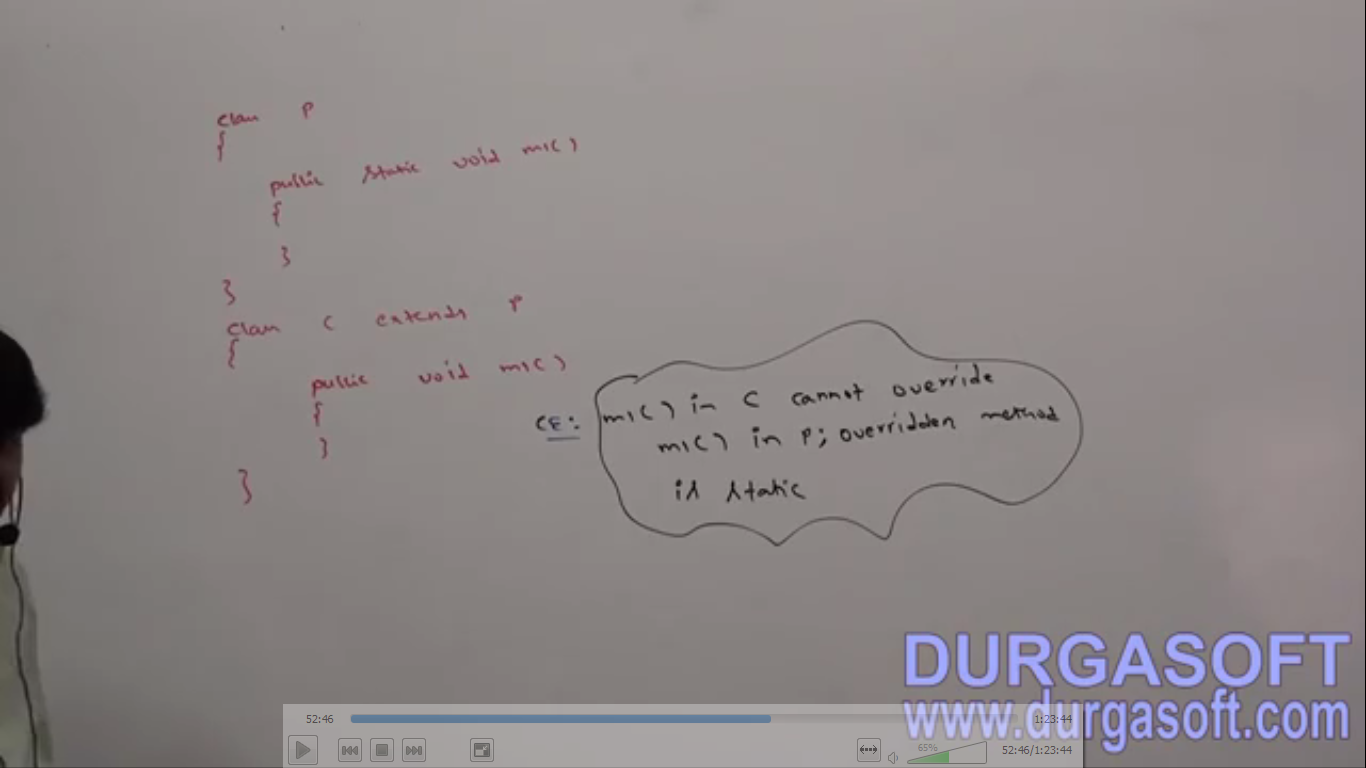
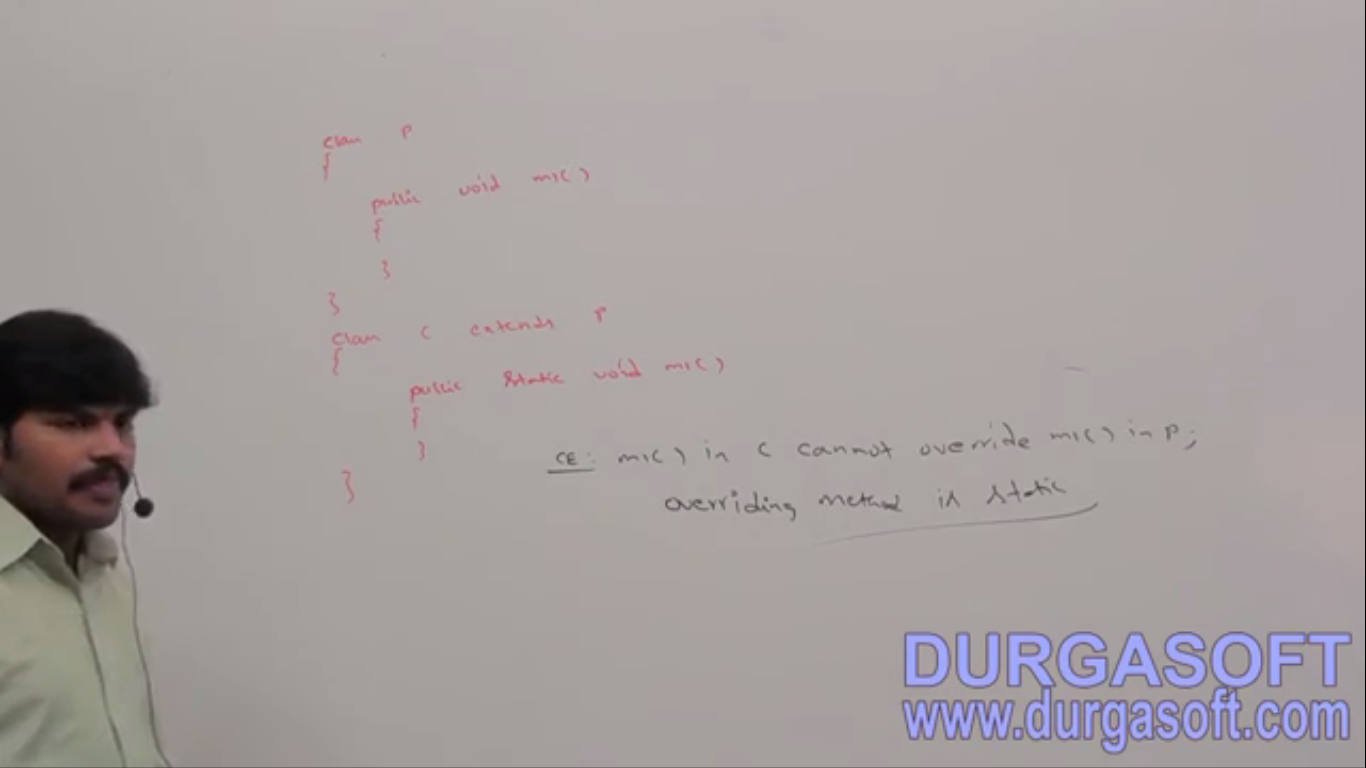
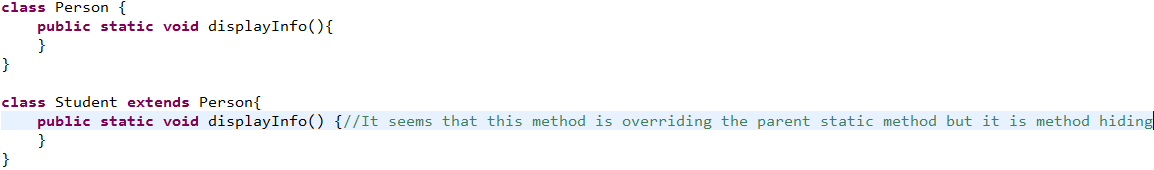
valid

valid

valid

|  |
| --- |
| Rule For Overriding |
| The method signature must be same in child class |
| Return type must be same till 1.4, co-variant type from 1.5 |
| Private method can’t be overridden |
| Final method can’t be overridden |
| Abstract method can be overridden |
| Can be declared to be Strictfp, synchronized, abstract, native place 🡺 no restriction |
| Scope (visibility) can’t be reduced but increased 🡺 Can’t be narrowed but widen |
| If child throws checked exception then parent must throw the same or its parent type. |

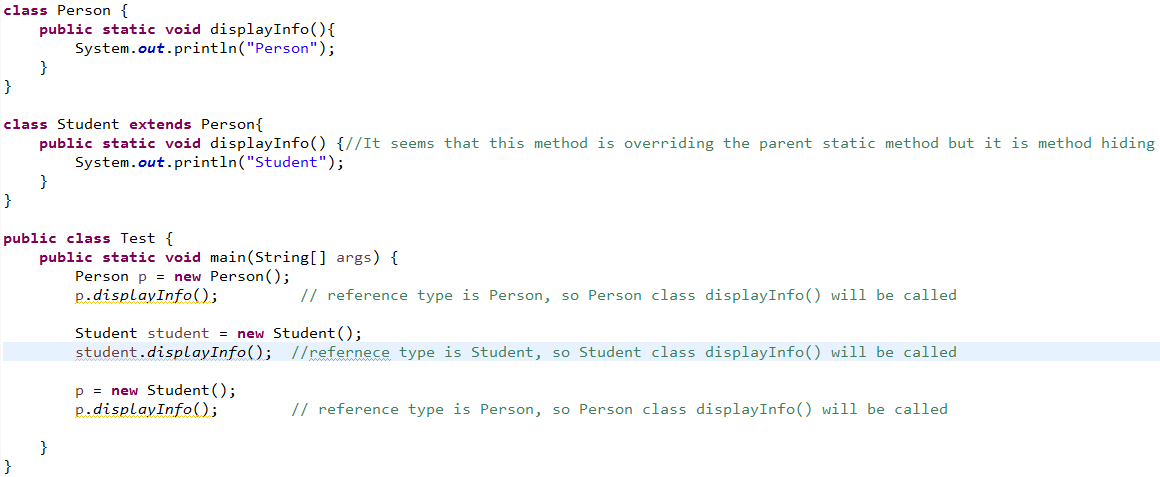
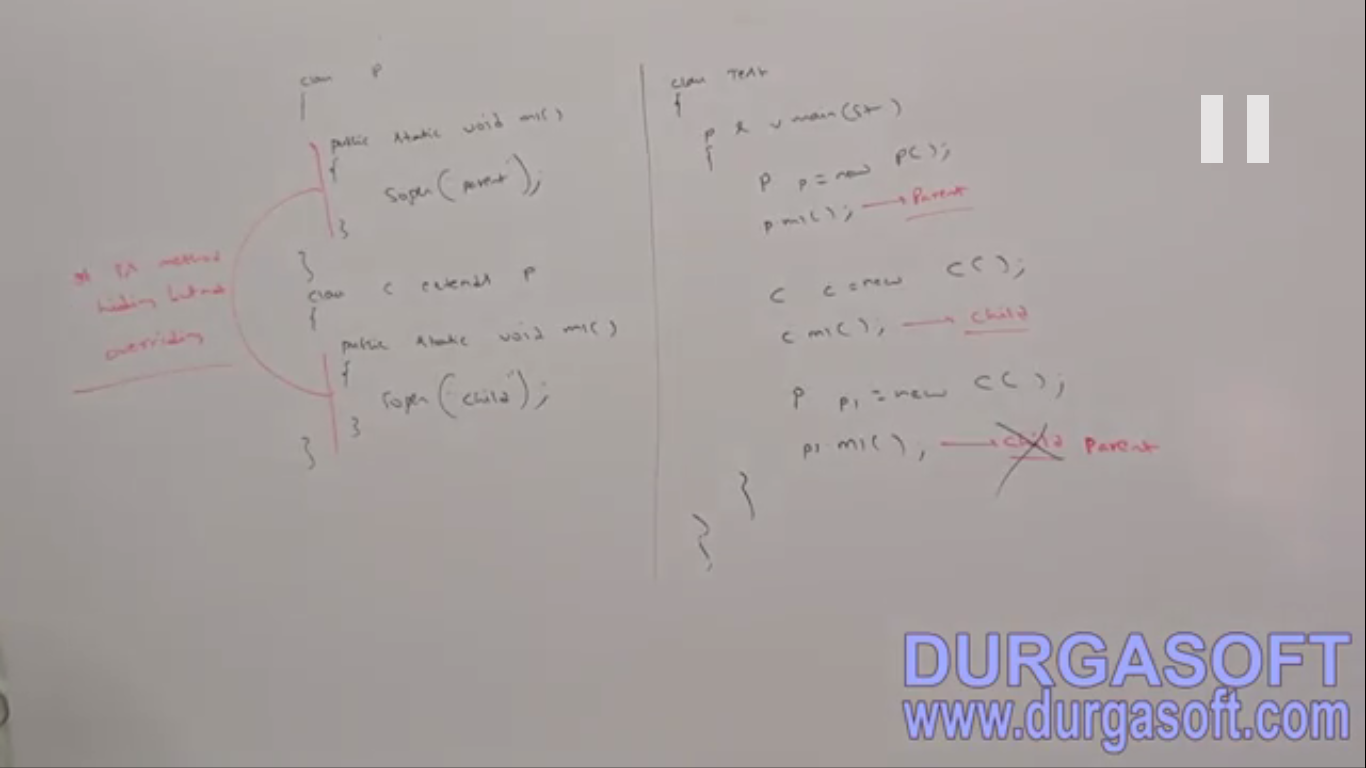
Overriding with respect to static method

1. Static method can’t be overridden with non-static method otherwise we will get compile time error.   
   **Scenario**: Can a faculty who left for better opportunity be fulfilled with a student or vice-versa.   
   
2. **Case 02**: Similarly, we can’t override a non-static method with static method.  
   
3. **Case 03**: Trying to overriding parent class static method with child class static method:  
   

If both the parent and child class method static with same signature, then we don’t get compile time error. It seems overriding concept applicable for static method. But it’s not overriding but it is method hiding.

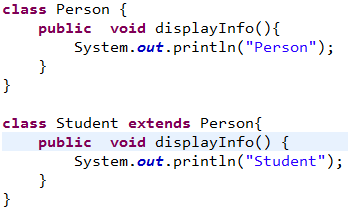
1. **Method Hiding:** All rules for method hiding are exactly same as those for overriding except the following differences:

|  |  |
| --- | --- |
| Method Hiding | Overriding |
| Both parent and child methods should be **static** | Both methods should be **non-static** |
| Compiler is responsible for method resolution based on  reference type. So runtime object has no effect. | JVM is responsible for method resolution based on  runtime object |
| Also know as  🡪 Compile-time polymorphism  🡪 Static Polymorphism 🡪 Early Binding | Also know as  🡪 Run-time polymorphism  🡪 Dynamic Polymorphism 🡪 Late Binding |

1. d

# Why overriding and hiding words are selected

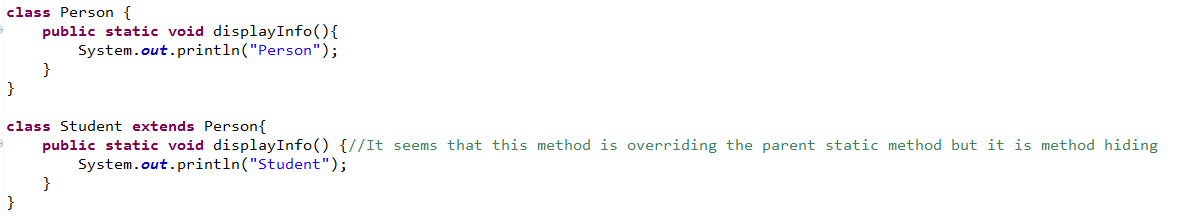
**Scenario**:   
**White-Board**: I write something on board then wipe then write something. It’s overriding. In overriding, the previous content is gone forever.   
**Chart**: I write something on chart then place a new chart on the previous chart and write something. If you ask for the previous content then I just move the front chart to show the backend chart. It is hiding. In hiding, the previous content is there.   
**NOW**: Technically   
**Overriding**:   


Person p = new Student();  
p.displayInfo();

Student s = new Student();  
s.displayInfo();

NOTE: here we are taking Parent class reference variable NOTE: Here we are taking Child class reference variable  
In both cases, child class displayInfo() will be called

**Hiding**:



Person p = new Student();  
p.displayInfo();

Here **output**: Person because reference type is Person

Student s = new Student();  
s.displayInfo();

Here **output**: Student because reference type is Student

NOTE: here we are taking Parent class reference variable NOTE: Here we are taking Child class reference variable  
So in hiding concept: we have the way to call either of the methods. Both copies of methods are available through Parent or child reference type which is not possible in case of overriding concept.