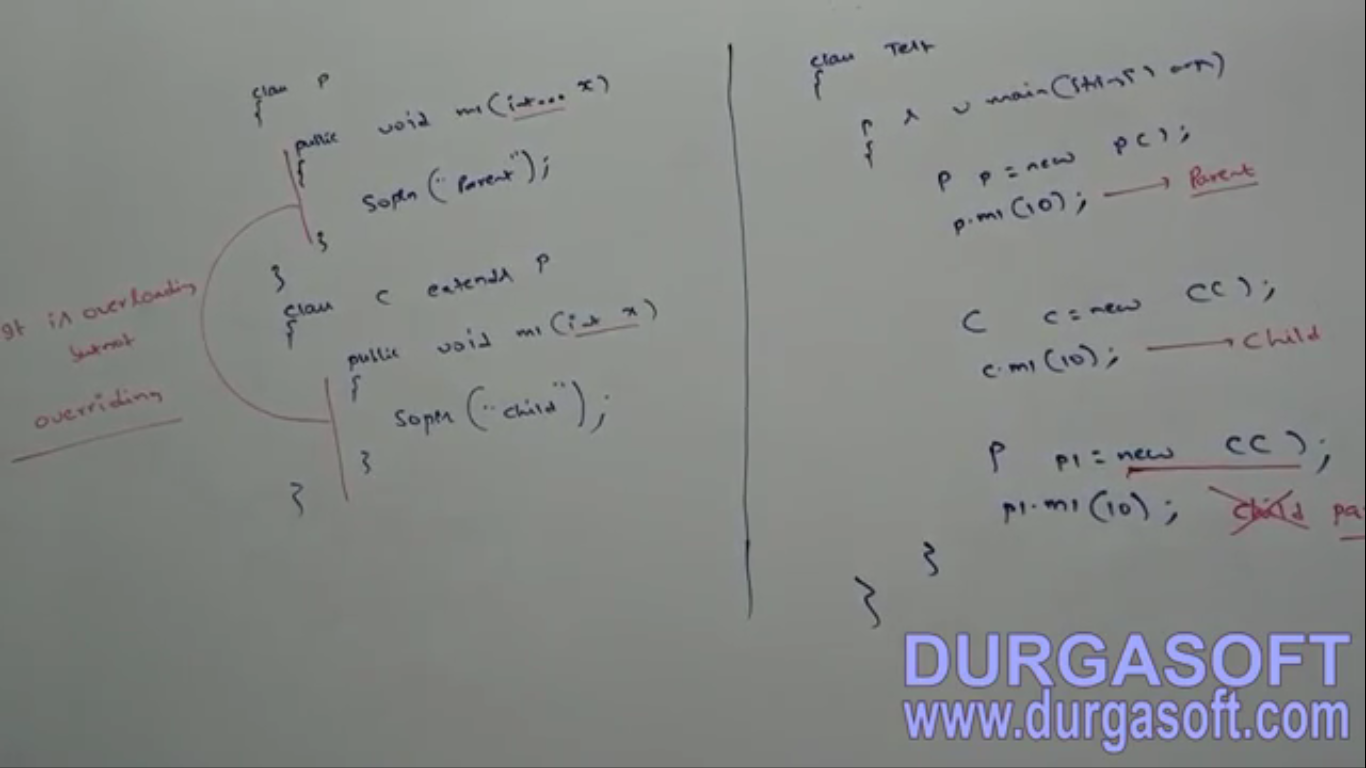
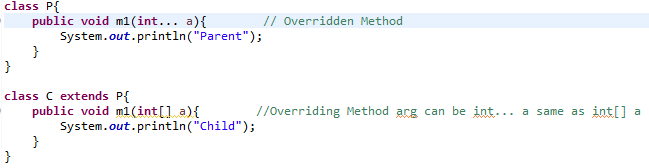
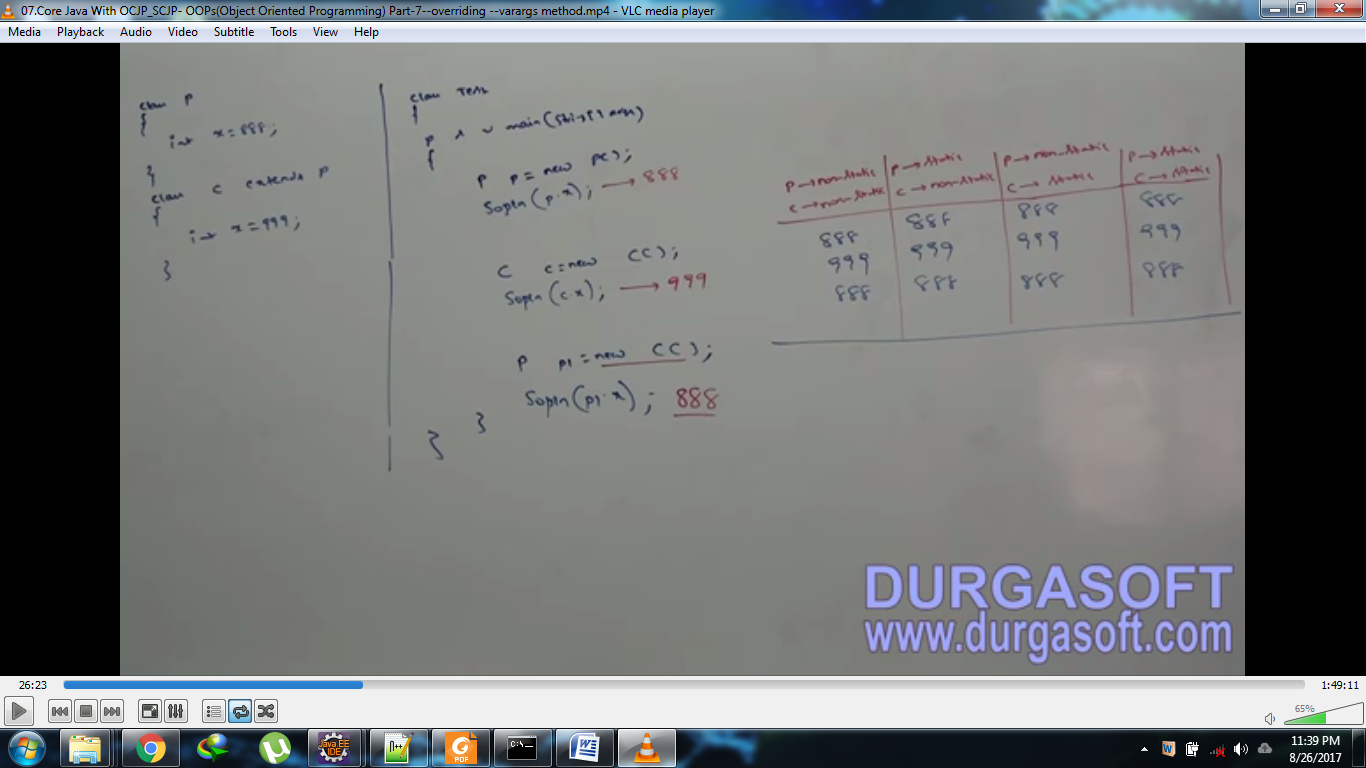
overriding with var-arg

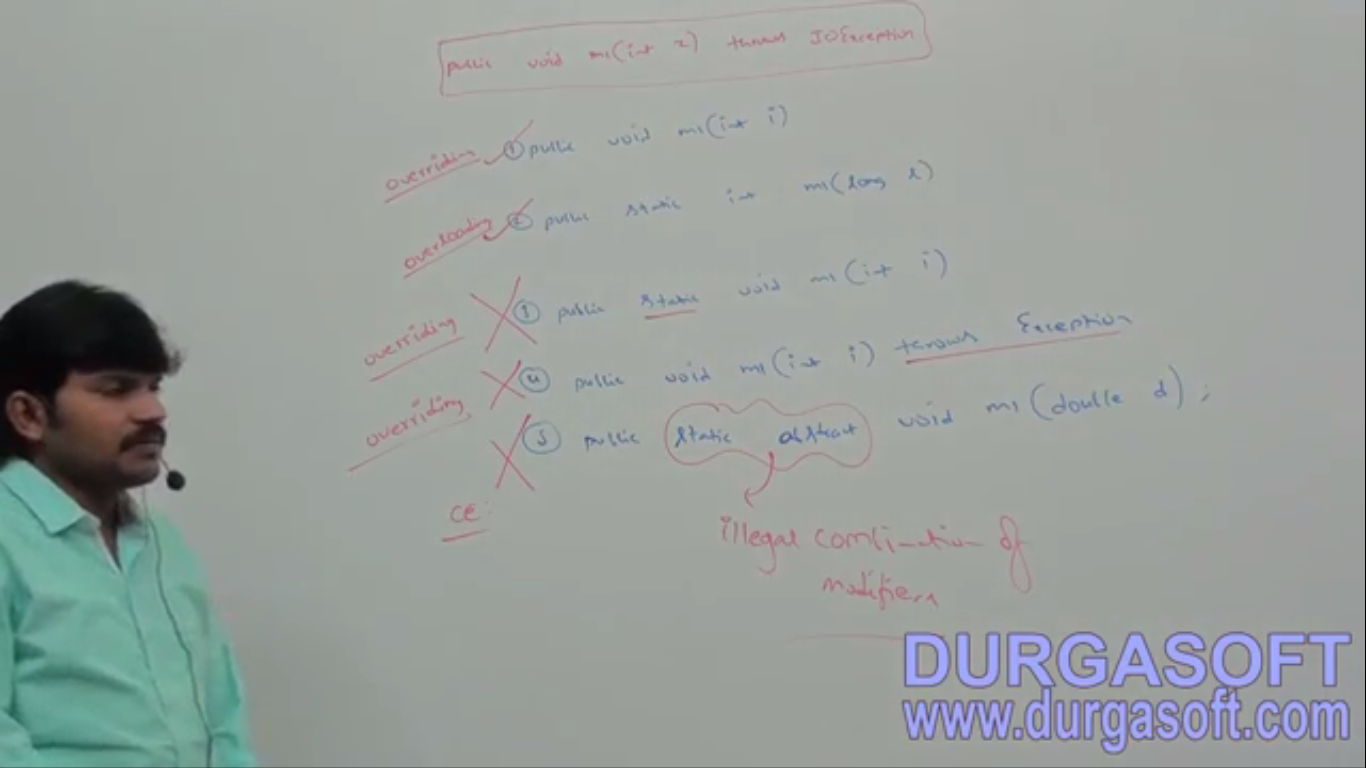
1. We can override var-arg with another var-arg method only. If we are trying to override with normal method, then it becomes overloading but not overriding.   
   
2. **Example  
   **

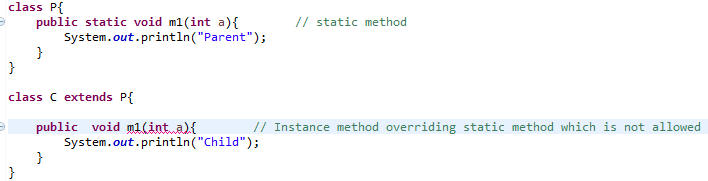
Overriding with respect to variable.

1. Variable resolution always takes care by compiler based on reference type irrespective of whether the variable is static or non-static. Overriding concept is applicable only for methods but not for variables.  
   **Jatin**: Variable resolution is **compile time polymorphism** 🡪 so resolved by compiler by reference type.   
   **NOTE**: No matter which combination of static and non-static in parent and child is used, rule is consistent.   
   

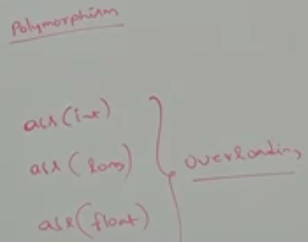
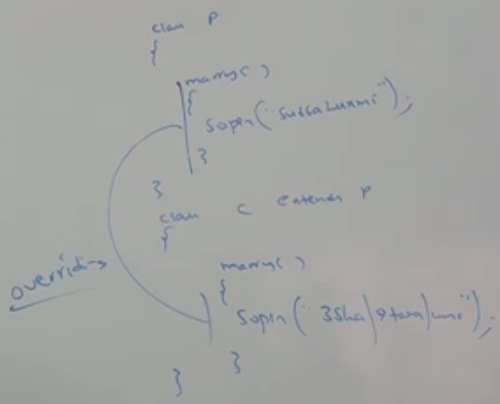
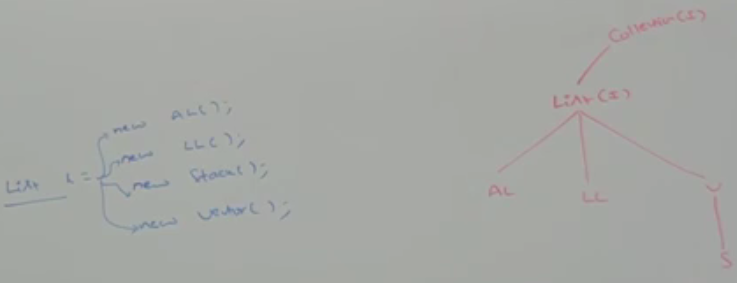
Difference b/w Overloading and Overriding

|  |  |  |
| --- | --- | --- |
| **Property** | **Overloading** | **Overriding** |
| Method Names | Same | Same |
| Argument types | Different[At least order] | Same including order |
| Method Signatures | Different | Same |
| Return Type | Not Applicable | Same[1.4], Same or Co-Variant[From 1.5] |
| Private, Static, final methods | Can be overloaded | Can’t |
| Access Modifiers | No Restriction | Can’t be reduced but can be increased |
| Throws | No Restriction | If child class method throws any checked Exception, compulsory parent class method should throw the same exception or its parent type but no restrictions for un-checked exception |
| Method resolution | **By compiler based on reference type** | **Always taken care of by JVM based on runtime object** |
| Also known as | Compile time poly, Static poly, Early Binding | Runtime Poly, Dynamic Poly, Late Binding |



**NOTE**: Instance method can’t override static method and vice versa.   


Polymorphism

1. One name but multiple forms is the concept of polymorphism.   
   **Example01**: Method name is the same but we can apply different types of arguments 🡪 **Overloading**  
     
   **Example02**: Method signature is same but in parent class one type of implementation and in child class different type of implementation🡪 **Overriding**  
    **Example03:** Using a parent reference to hold child object is the concept of polymorphism.   
   
2. Parent class reference to hold child object but by using that reference we can call only methods available in parent class and we can’t call child specific methods.   
   BUT!!! By using child reference we can call both parent and child class methods.   
   **When we should go for parent reference to hold child object**?  
   If we don’t know the exact runtime type of object then we should go for parent reference.   
   **For Example**: 1st element present in the ArrayList can be of any type. Hence the return type of get() is Object which can return any type of object. **Object o = l.get(0)**;

|  |  |
| --- | --- |
| **C c = new C()** | **P p = new C();** |
| When exact runtime type of object is known | When not known |
| Both parent and child class methods can be called | Only parent specific methods |
| Can be used to store only child object reference | Can be used to hold parent and any child object reference |

1. 