overloading cases

**\*\*\*\*NOTE: Always remember these points**

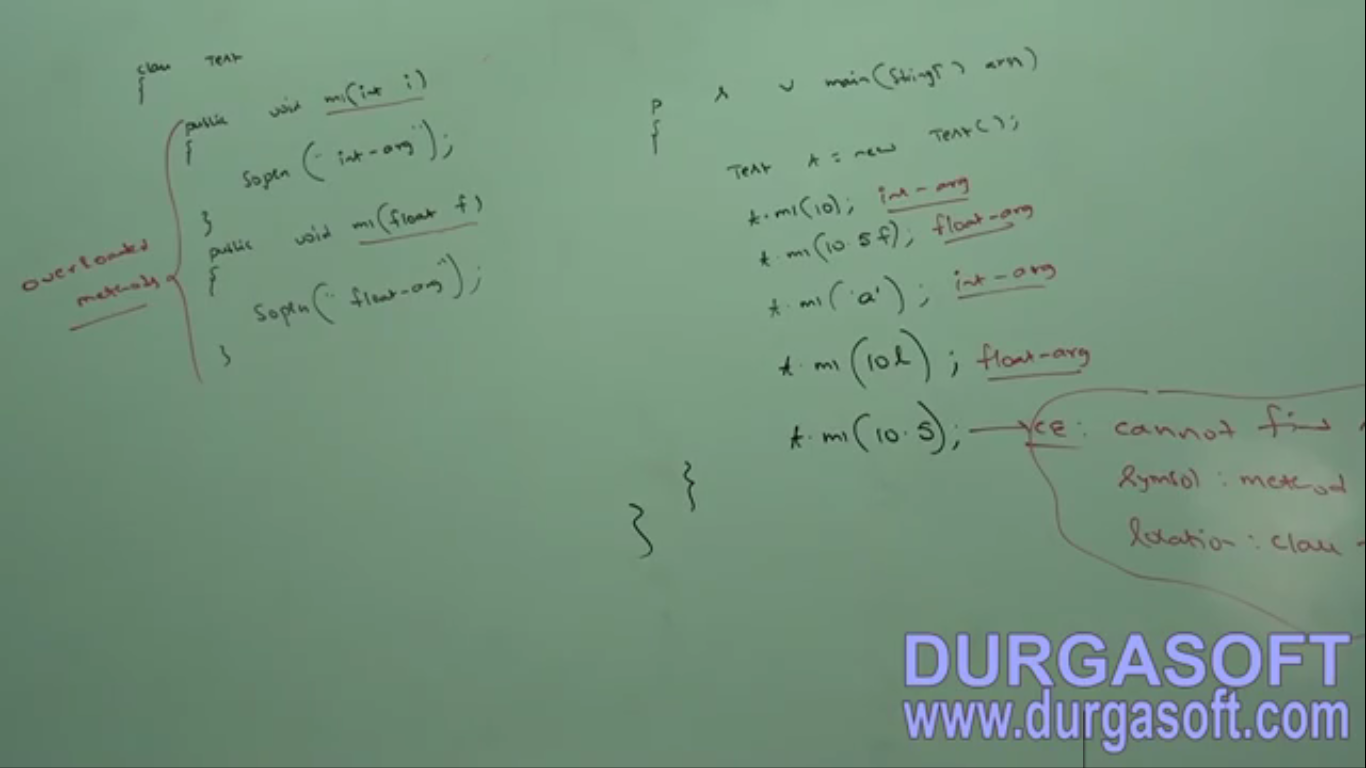
1. Method binding is done at compile time when there is no object created yet. So object itself plays no role in binding. So in case of reference type, reference type plays the role to help compiler to decide which signature, this method call matches.   
   **Suppose we are calling** 🡪 m1(person) where we have stored student into person reference variable. But as Student has not been yet created at compile time when compiler is binding method call to method definition, so compiler checks the type of reference variable and accordingly takes decision. Here method which takes argument of person type will be called.
2. When there is no exact match and compiler now has to promote the argument types. In this case, each method will be checked that after promotion, that method call matches if yes, match is found. In this way more than one match can be found hence ambiguity problem.
3. If method with old concept and a method with new concept match then old concept method wins to provide compatibility.
4. **Case01 Automatic Promotion in overloading**
   1. While resolving overloaded methods, if **exact match method** is not available, then we will not get any compile time error immediately. 1st compiler will promote argument to the next level, and check whether matched method is available or not. If matched method is available then it will be considered and if the matched method is not available then compiler again promotes to the next level. This process will continue till all level. If matched method is not available, then we will get compile time error. The followings are all **possible promotions** in overloading.

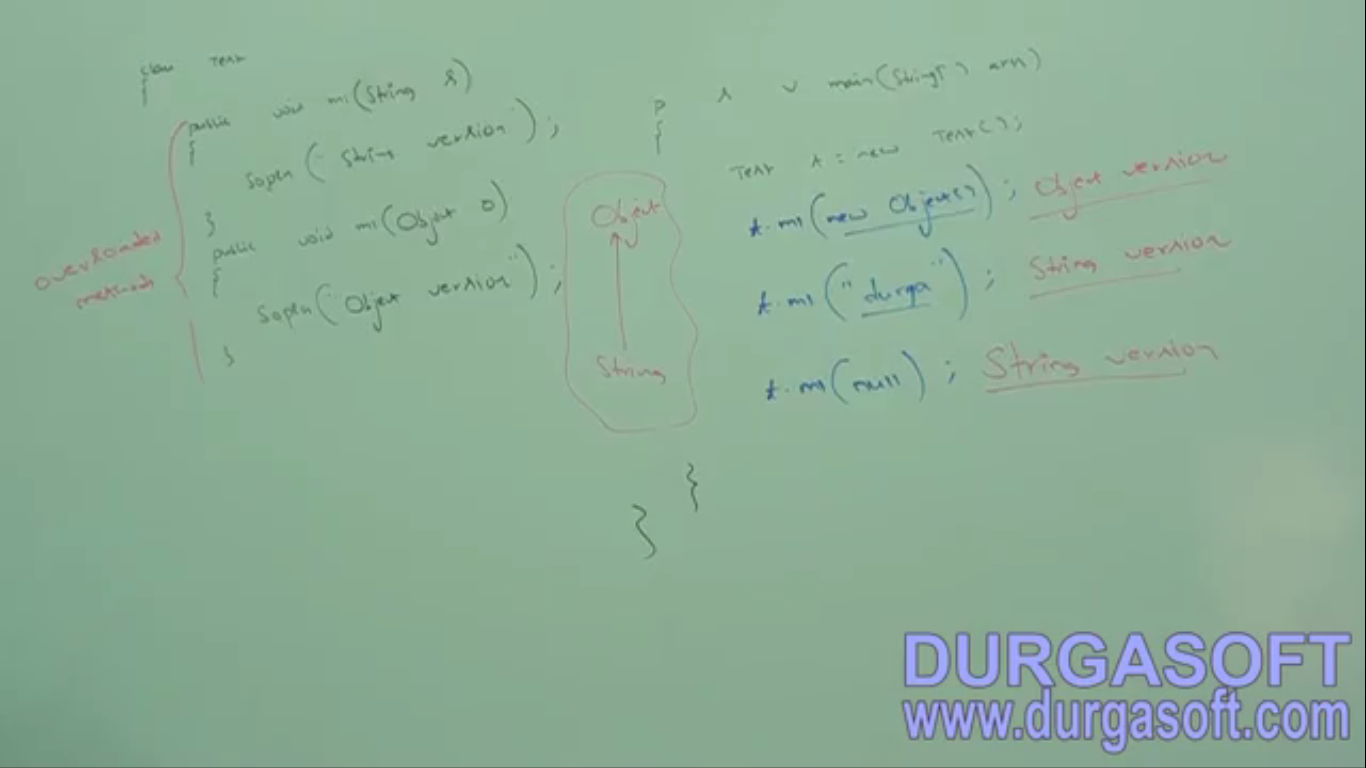
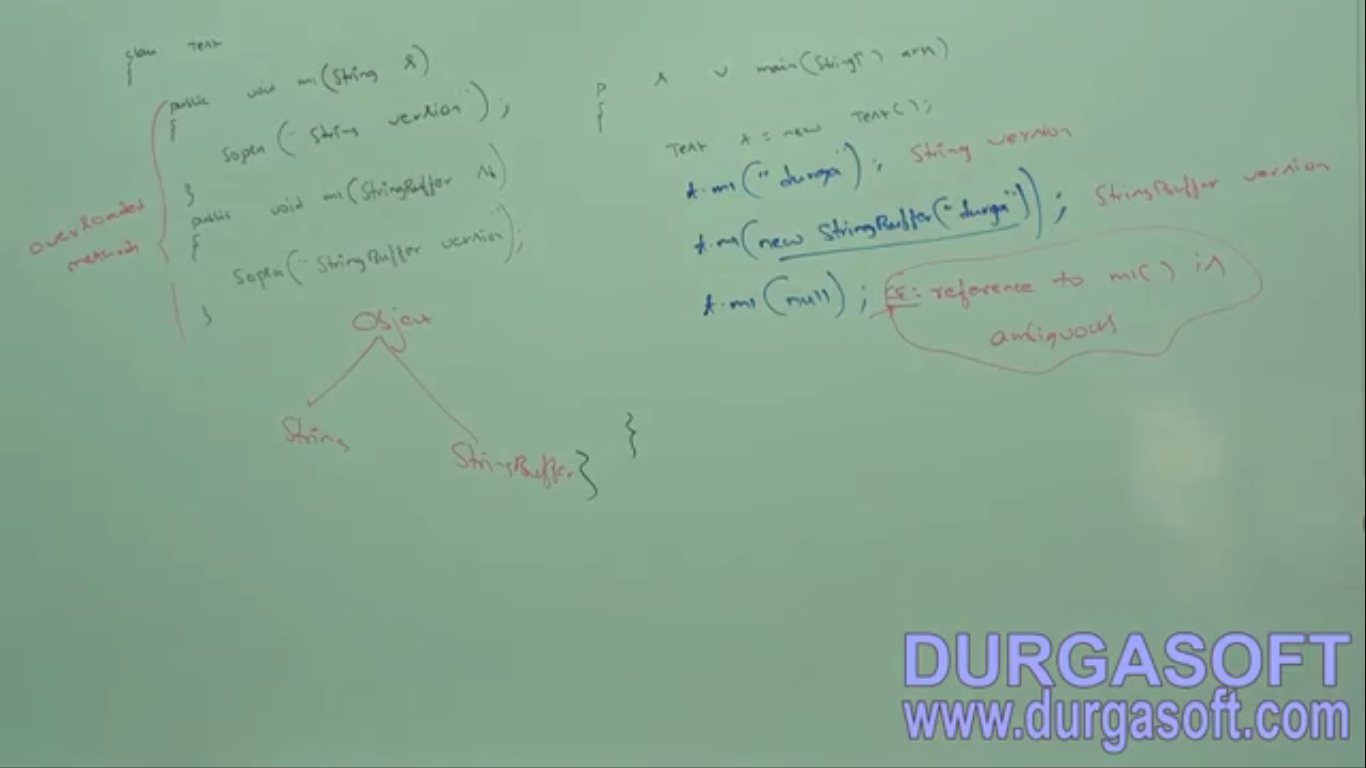
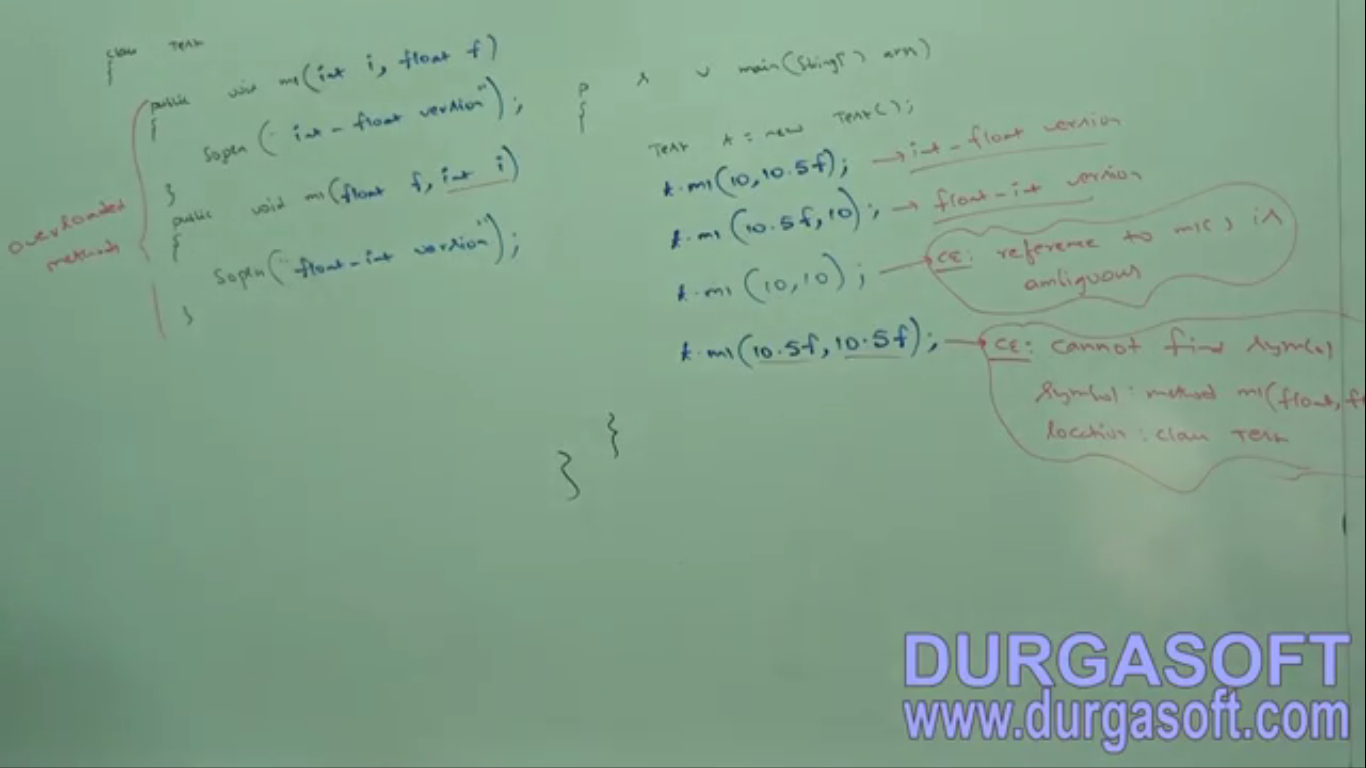
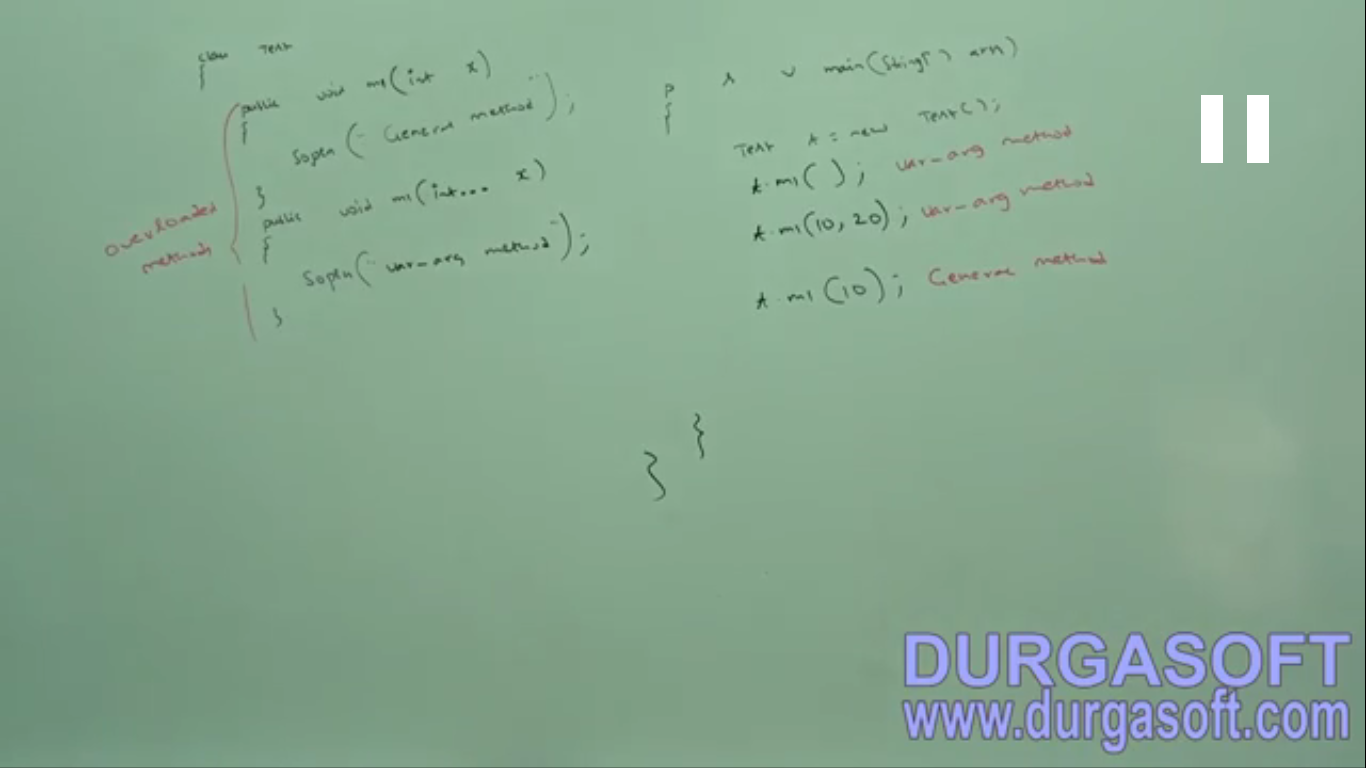
int 🡪 long 🡪 float 🡪 double

byte 🡪 Short

char

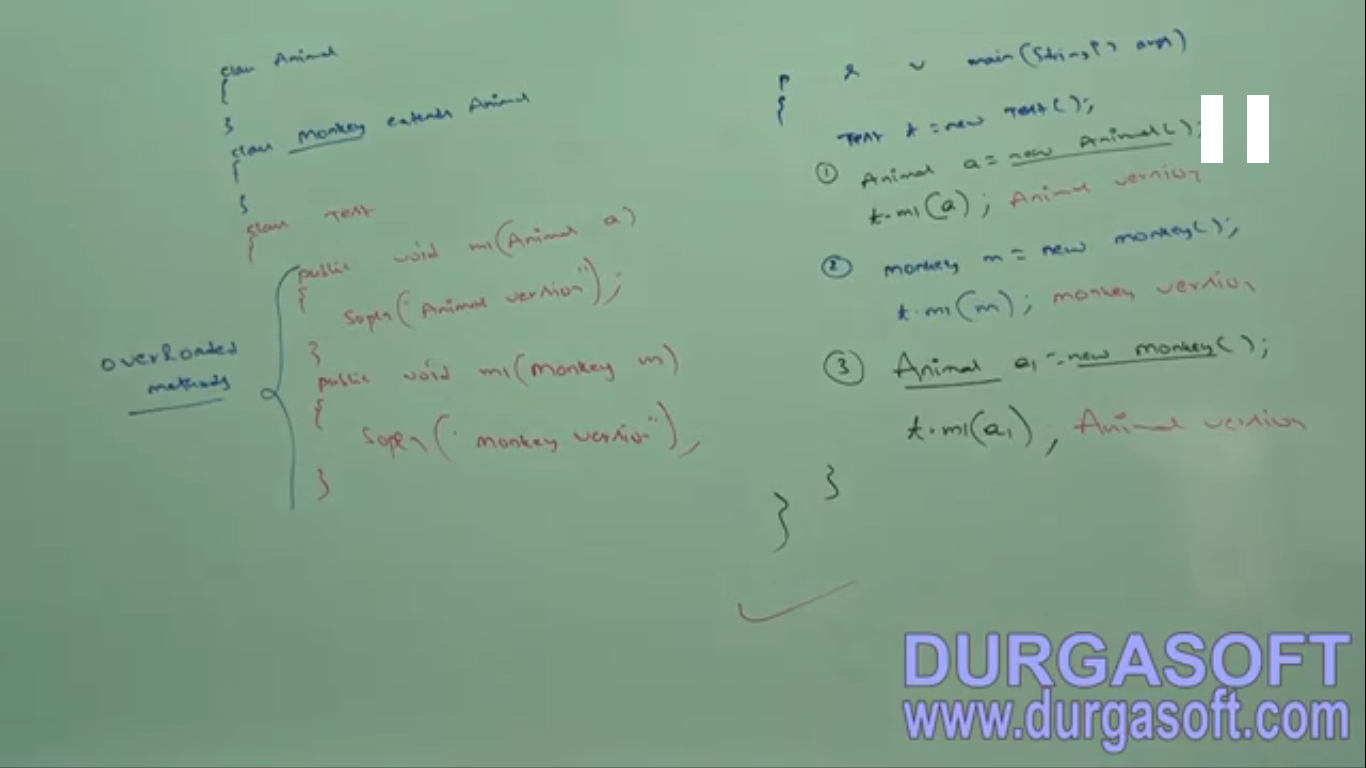
**Automatic Promotion in Overloading**

**Example**:  


1. **Case02**: While resolving overloaded methods, compiler will always keep the precedence for child type when compared with parent type argument.   
   
2. **Case03**: if we are calling a method passing null, and there are methods more than one applicable and there is no parent-child relationship in arguments of those methods in that case compiler will not be able to resolve and will result in ambiguity. 
3. **Case 04:**   
   **Jatin🡪 ambiguity**: When more than 1 method match after promotion of passed arguments🡪 **ambiguity  
   Jatin🡪 Can’t find symbol:** When neither exact match, nor match with promotion nor ambiguity.   
   
4. **Case 05**: When there is fight b/w old concept and new concept, old concept always wins to provide compatibility with the old concept.   
     
   var-arg argument method has least priority.

This is new concept introduced in 1.5v

old concept

1. **case 05**:   
     
   In overloading, method resolution is always taken care of by compiler based on reference type. In overloading, runtime object will not play any role.