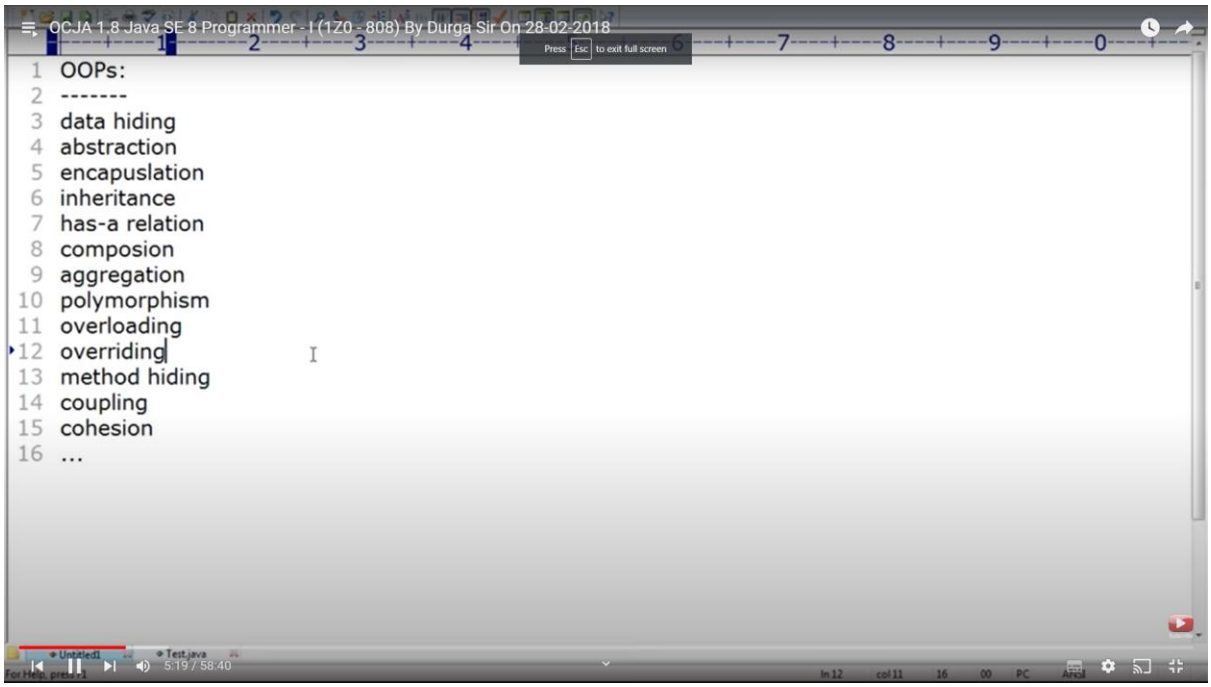
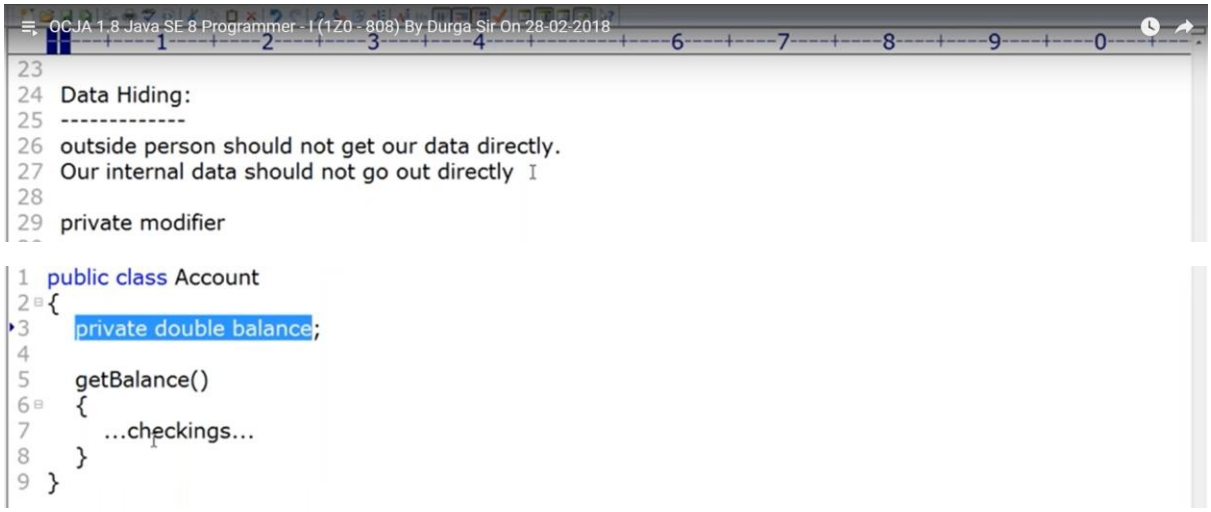
OOPS

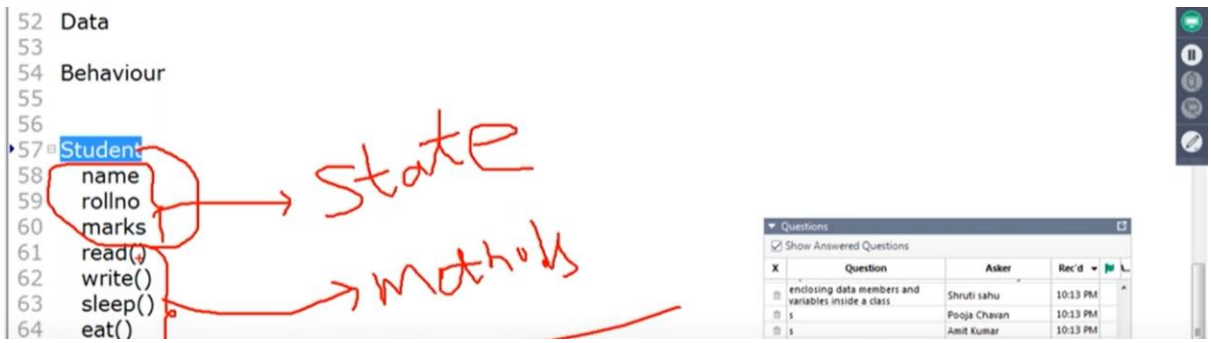





```
public class Account
{
   private double balance;

   getBalance()
   {
      ...checkings...
   }
}
```

Process of binding data and coressponding methods into single unit is called encapsulation

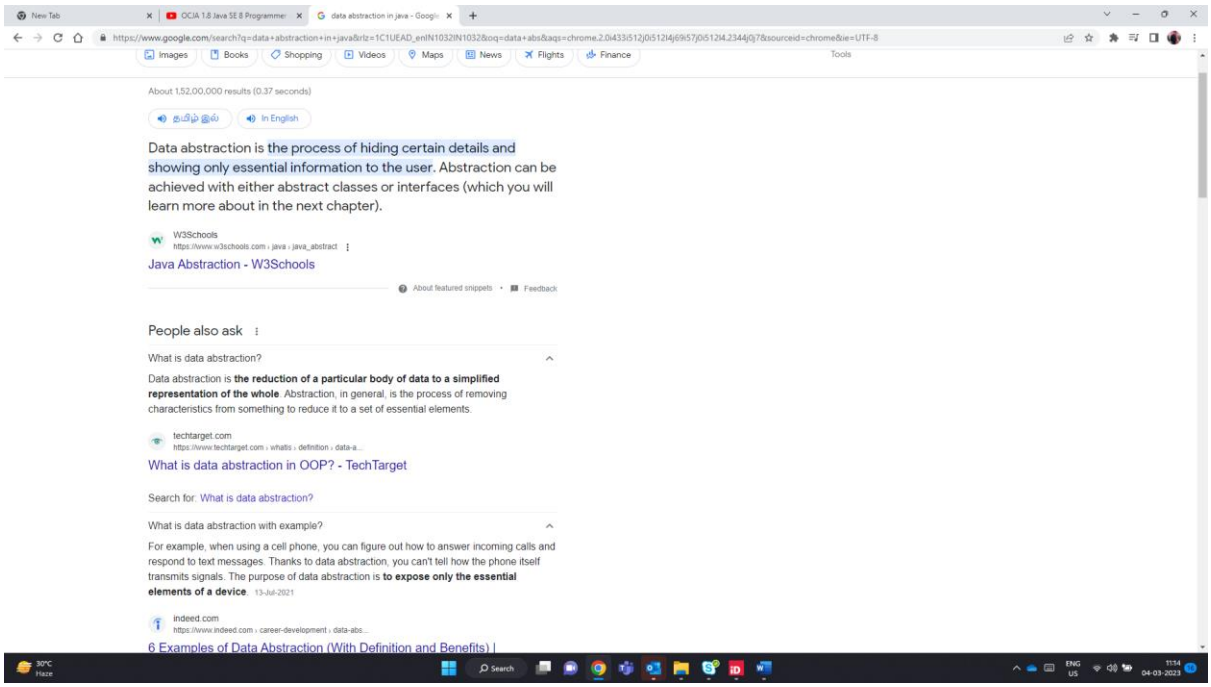
Variables are states (values it takes) methods are behaviours



```
68  Data Hiding and Abstraction...Encapsulation
69
70  Encapsultation=Data Hiding+Abstraction
```

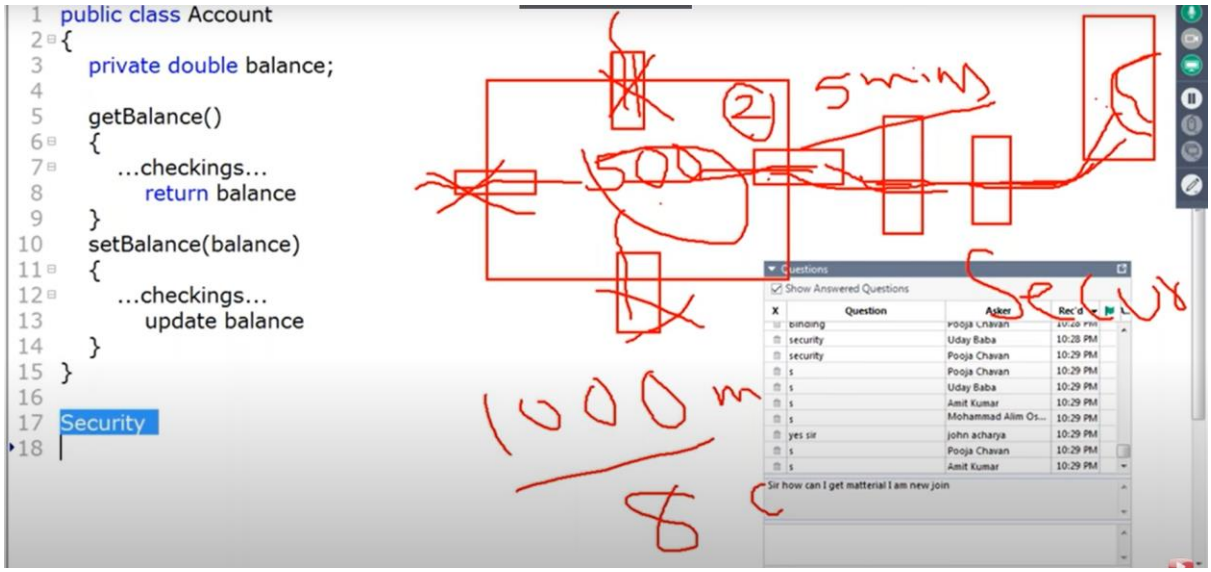
Data hiding is here. Where is absraction. Here the calculations parts , all complex calculations are hidden from the user and we expose only the methods which are needed , this is called abstraction

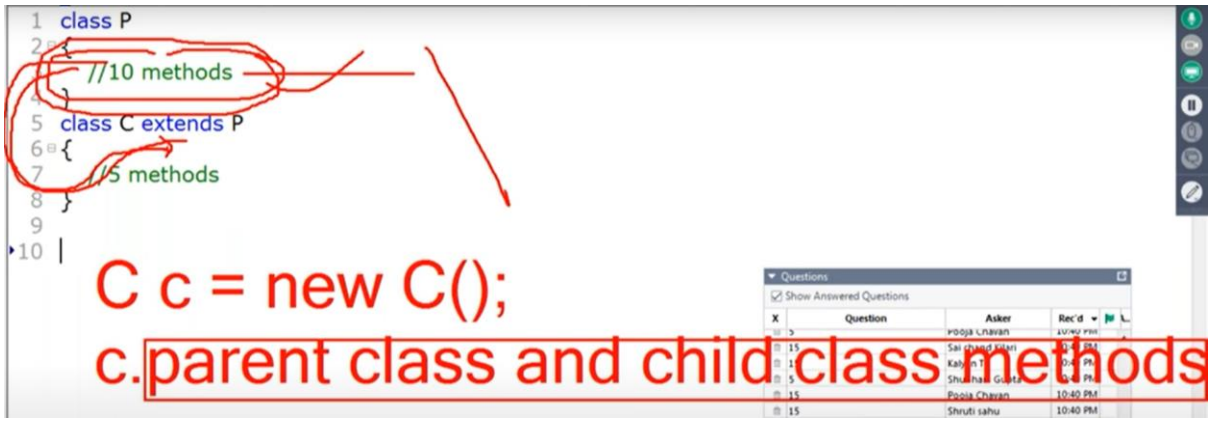
```
1  public class Account
2  {
3     private double balance;
4
5     getBalance()
6     {
7        ...checkings...
8           return balance
9     }
10    setBalance(balance)
11    {
12       ...checkings...
13          update balance
14    }
15 }
```



Security – should comprimise performance

Limit access. Getter setter authorization aunthentication etc. slows execution





Reusabiltity

```java
class P
{
   public void m1()
   {
      System.out.println("Parent Method");
   }
}
class C extends P
{
   public void m2()
   {
      System.out.println("Child Method");
   }
}
class  Test
{
   public static void main(String[] args)
   {
      C c= new C();
      c.m1();
      c.m2();

   }
}
```

Inheritance(IS-A Relationship)

for code reusability and to extends existing functionality with some extra functionality

1. What ever methods present in parent class are by default available to the child class.Hence on the child reference we can call the methods present in both parent class and child class

```java
      P p = new P();
      p.m1();
      //p.m2();
```

```
D:\durgaclasses>javac Test.java
Test.java:25: error: cannot find symbol
                p.m2();
                  ^
  symbol:   method m2()
  location: variable p of type P
1 error
```

The below is right parent ref child object. But why

```java
      P p = new P();
      p.m1();
      p.m2();*/

      P p = new C();
      p.m1();
      p.m2();
```

polymorphism

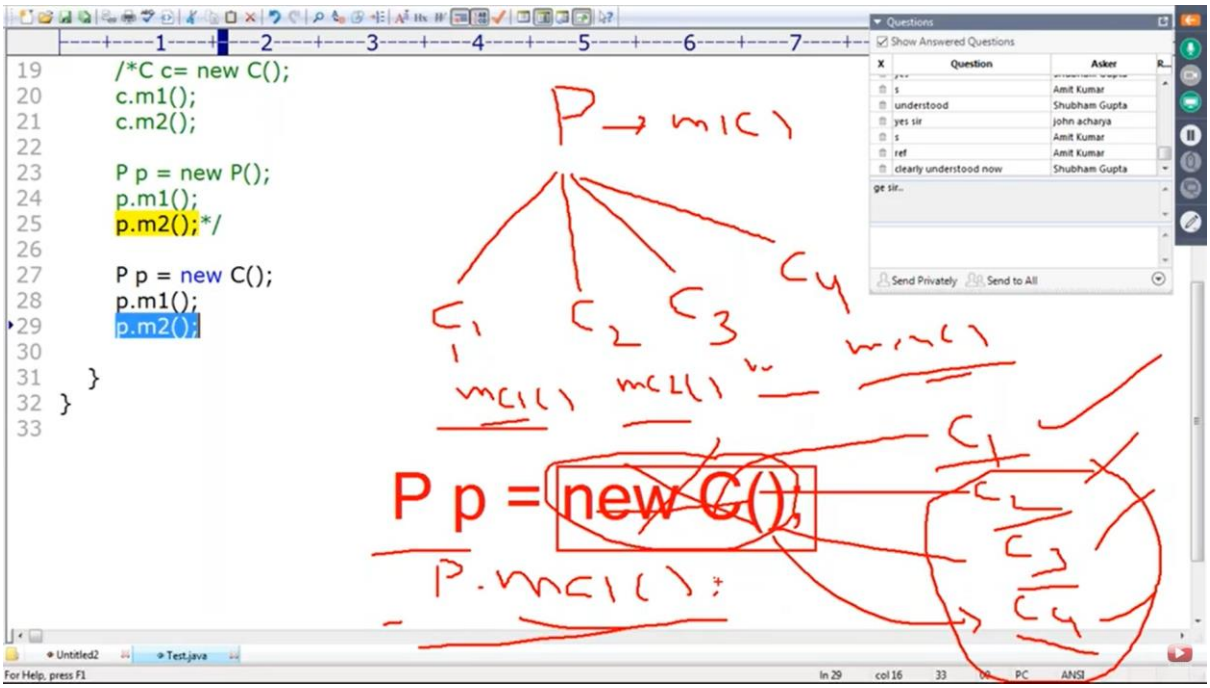
compiler and jvm==>
compiler will always check syntax
jvm will always runs the code

compiler will always consider reference type but not runtime object
but jvm will always consider runtime object but not reference type

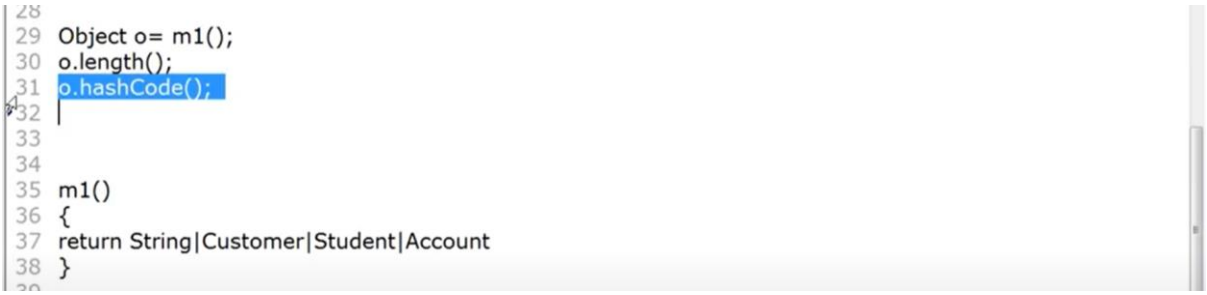
Compiler never checks below

Here compiler will not accept p.m2 because

P can have any no of child class and they would have their own imp of m2(). Compiler does not know what object p holds only at run time JVM knows and hence compiler does not accepts this .

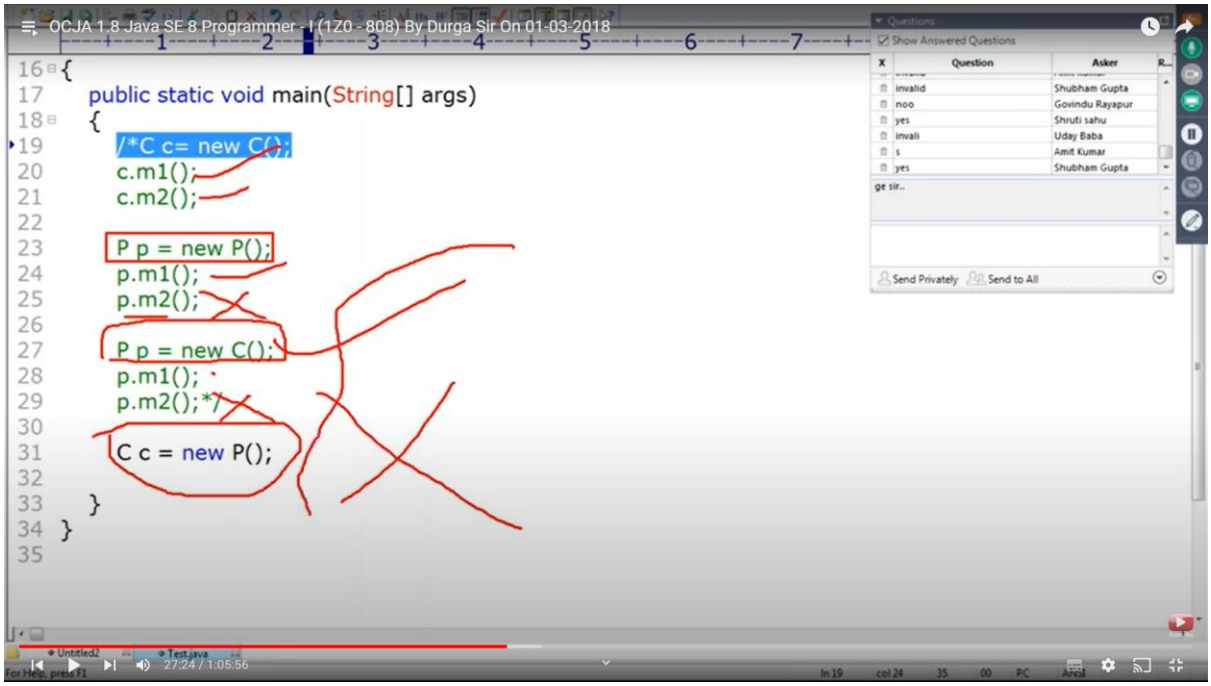


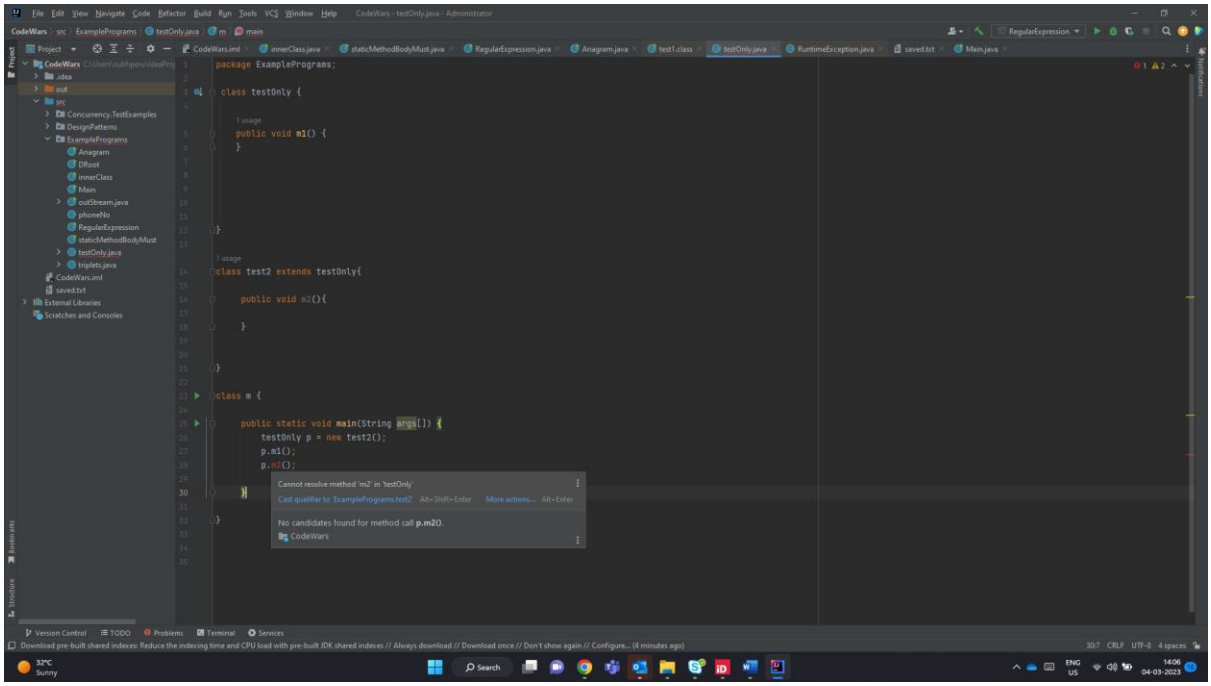
Here 0.length() throws error , because o is of object type. Parent reference can use any parent specific methodd. Parent methods are available for any child class. But child methods are not avaiable for any Parent reference. Hence u cant call o.lenth() which is specifc to child class String whereas o.hashcode() specific to parent can be used

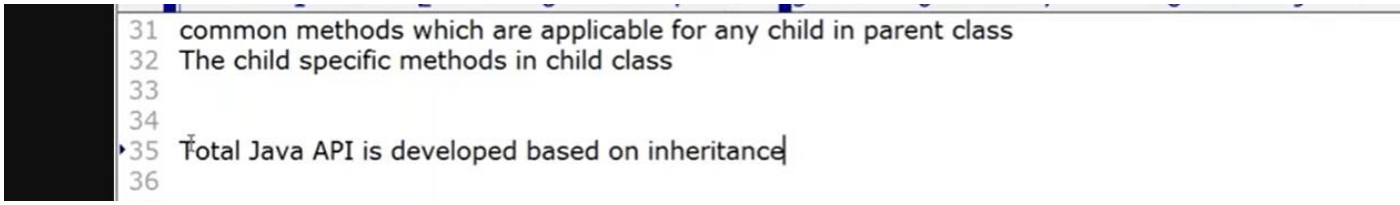


Parent ref can be used to hold child objects wheras child ref cannot parent objects. Below is wrong

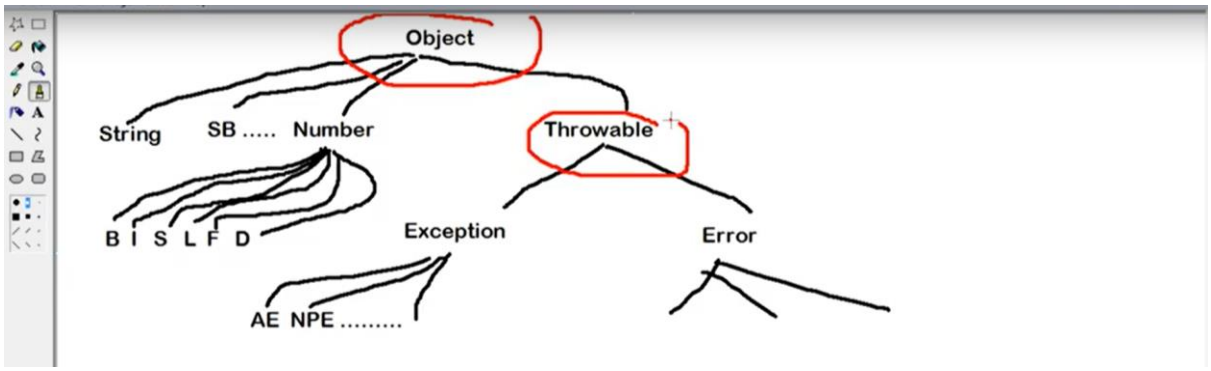
```
30
31      C c = new P();
32
```



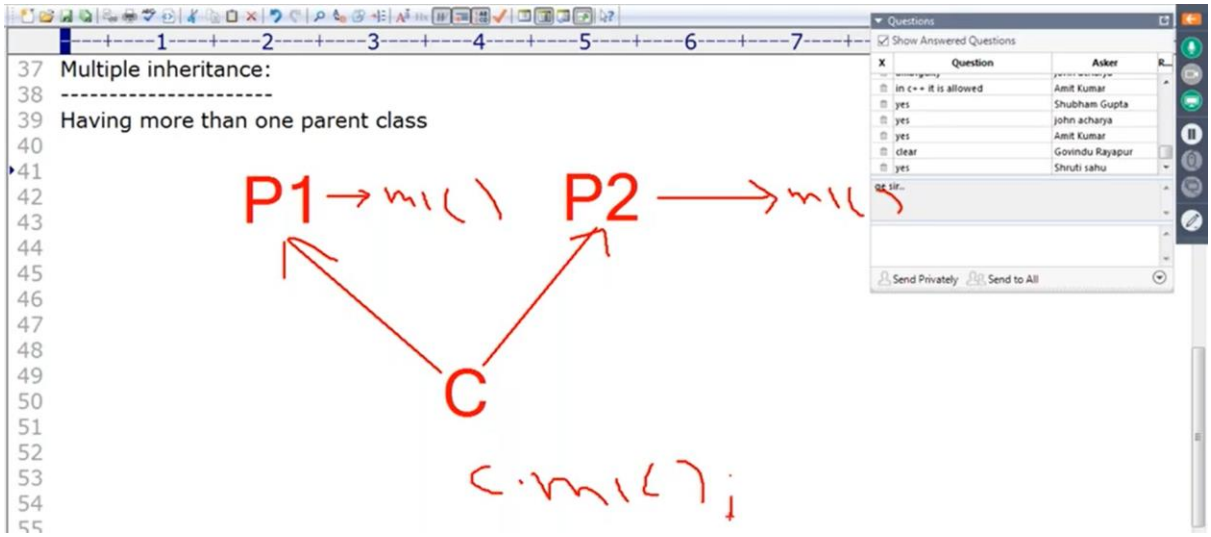


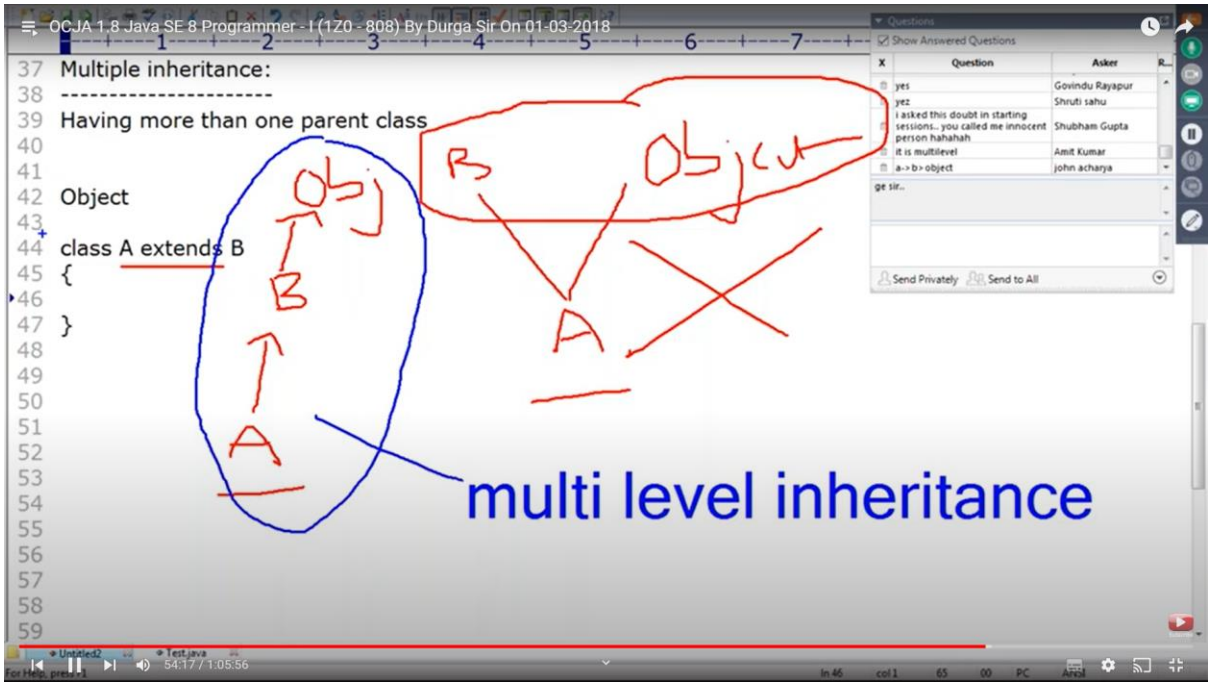


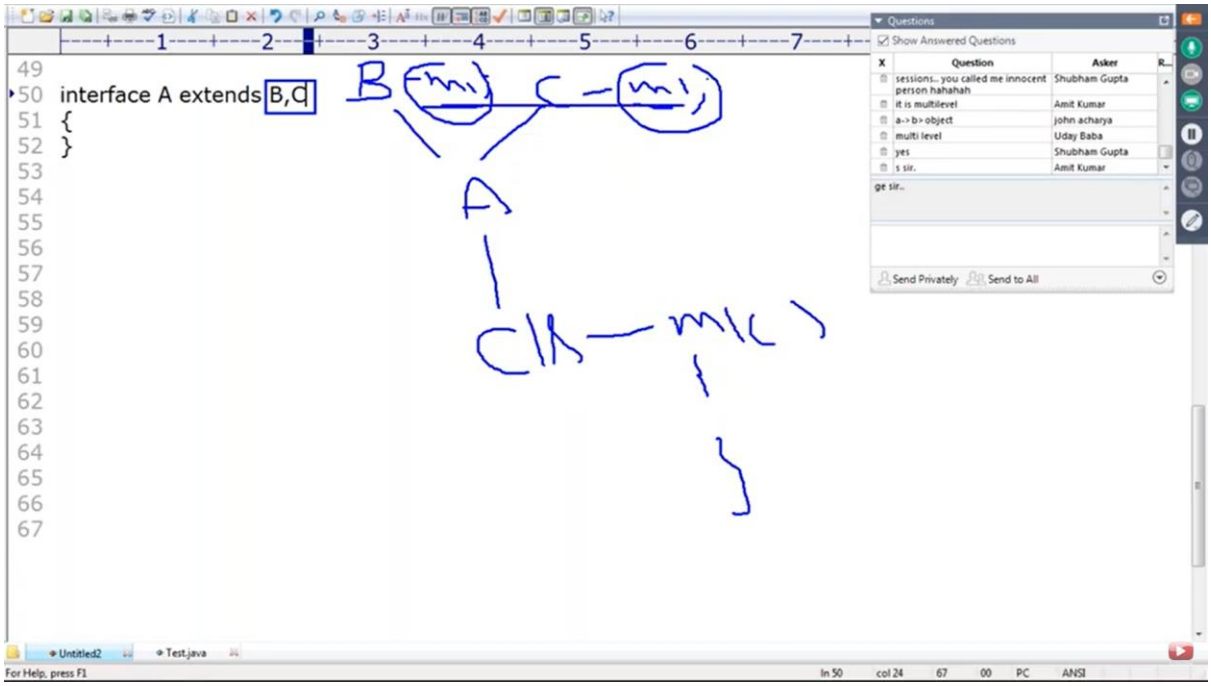
The most common methods which are applicable for all objects are available in Object class. The most common methods applicable for excpetions and errors are available in throwable



Why class cannot extend multiple class at the same time. Ambiguity. Multiple inheritance is not possible in Java







One implementation is enough. Ambiguity is not there in interface

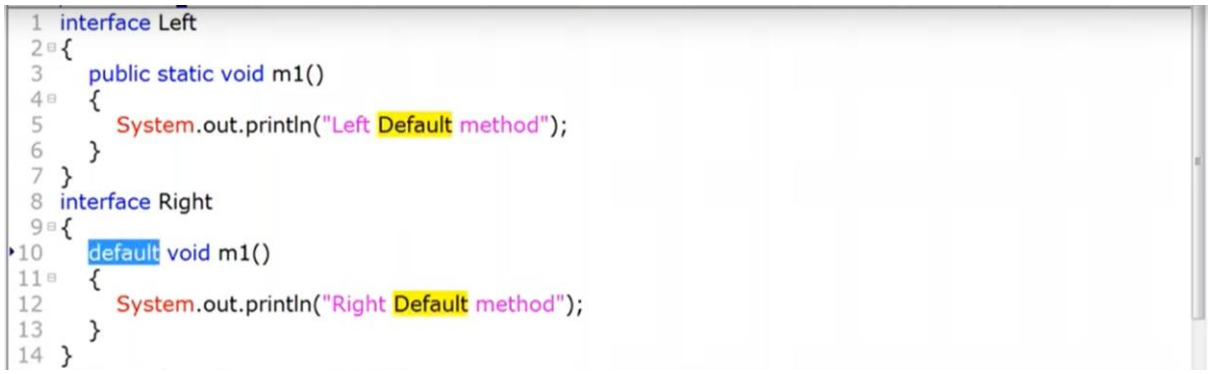
```java
interface Left
{
    default void m1()
    {
        System.out.println("Left Default method");
    }
}
interface Reft
{
    default void m1()
    {
        System.out.println("Right Default method");
    }
}
class  Test implements Left,Right
{
    public static void main(String[] args)
    {

    }
}
```

You can avoid this by overriding the method m1 in child class like below

```java
class  Test implements Left,Right
{
    public void m1()
    {
    }
    public static void main(String[] args)
    {

```

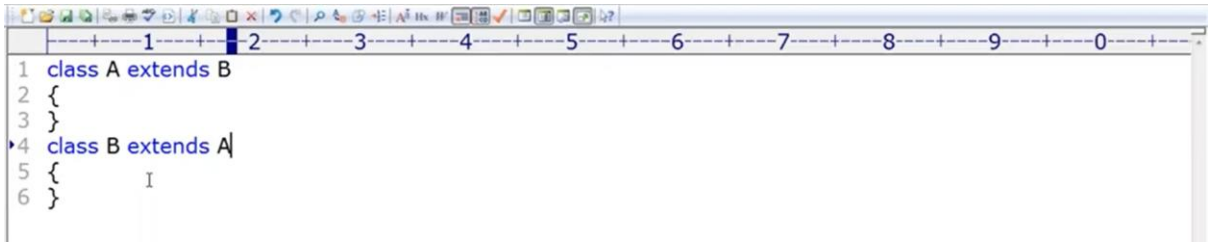
```java
class  Test implements Left,Right
{
    public void m1()
    {
        Left.super.m1();
        Right.super.m1();
    }
    public static void main(String[] args)
    {
        Test t = new Test();
        t.m1();
    }
}
```

Interface static method is not available to implement class its available to all should be called with interface name. Interf.staicmethoc()

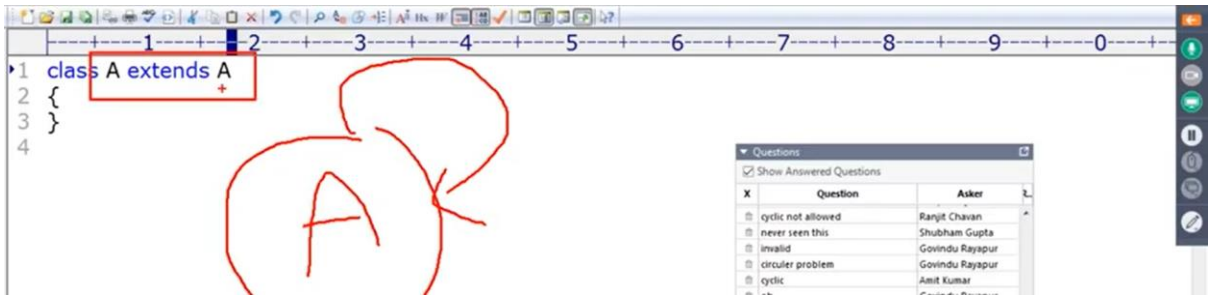


```
interface Left
{
   public static void m1()
   {
      System.out.println("Left Default method");
   }
}
interface Right
{
   default void m1()
   {
      System.out.println("Right Default method");
   }
}
```

cyclic inheritence- Not allowed – Cyclic inheritence



```
class A extends B
{
}
class B extends A
{
}
```



```
class A extends A
{
}
```


Questions

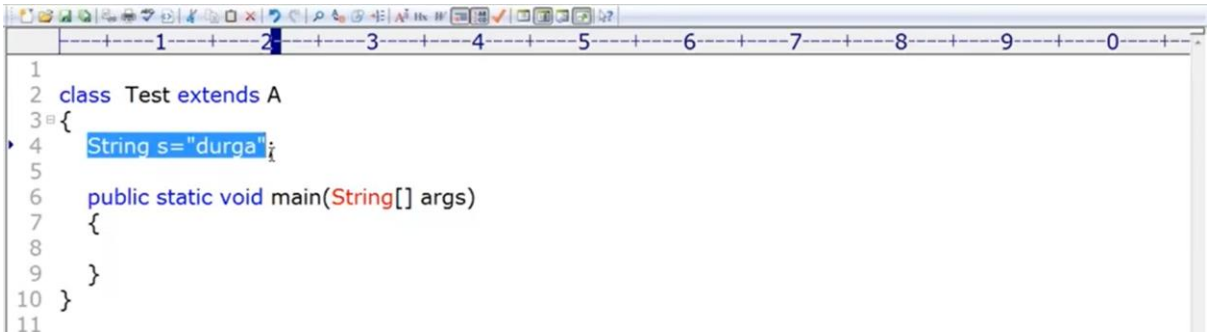
Show Answered Questions

X	Question	Asker
	cyclic not allowed	Ranjit Chavan
	never seen this	Shubham Gupta
	invalid	Govindu Rayapur
	circuler problem	Govindu Rayapur
	cyclic	Amit Kumar

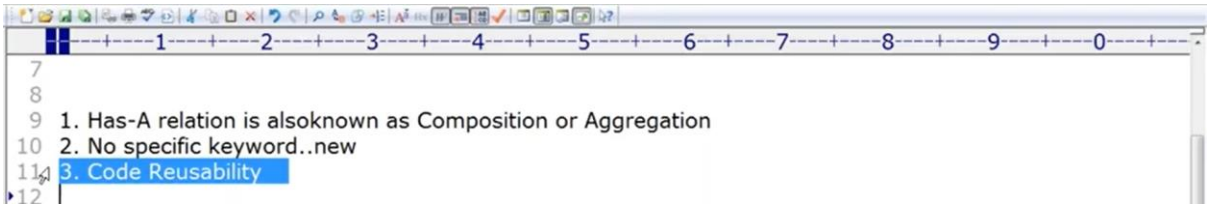
HAS A RELATION

If a class extends a class – Is A Relationship

If a class uses another class ref – line 4 Has a Relationship. Higly used relation



```
1
2  class  Test extends A
3  {
4     String s="durga";
5
6     public static void main(String[] args)
7     {
8
9     }
10 }
11
```



```
7
8
9  1. Has-A relation is alsoknown as Composition or Aggregation
10 2. No specific keyword..new
11 3. Code Reusability
12
```

If you want to extend the functionality use Is A relation

If you want to use a functionality of a class use Has A relation. Use someother class

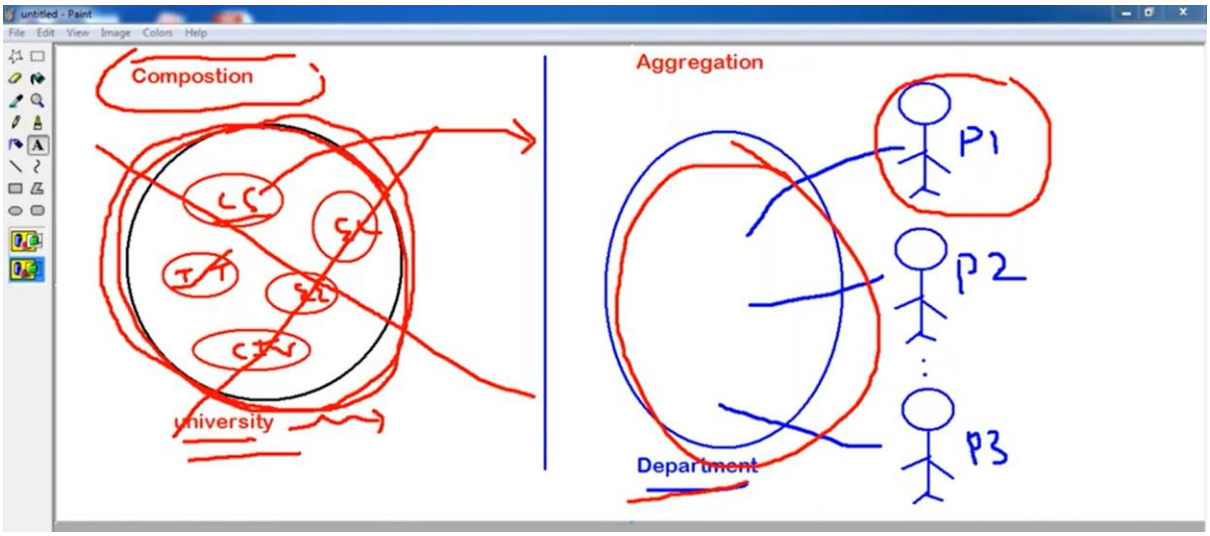
Has a relation

Has a Relation

Composition and Aggregation

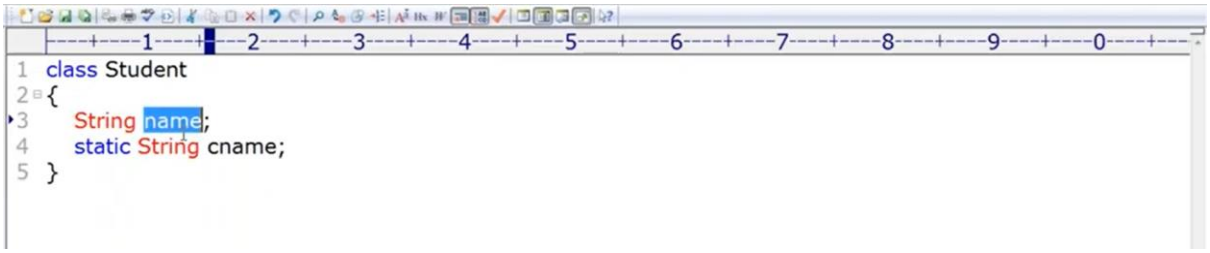
If the contained objects are strongly associated with the container object it's called Composition.

If there is a week association then its called Aggregation



In composition , the contained objects are created inside the container object whereas in Aggregation the contained objects are outside the container object and accessed by reference

Both are has a , without name there could not be a student strong association , without college name there can be student. So there is a week association between student and cname



```
1 class Student
2 {
3    String name;
4    static String cname;
5 }
```

Composition is a strong type of **"has-a"** relationship because the containing object is its owner. So, objects are tightly coupled, which means if we delete the parent object, the child object will also get deleted with it.

256

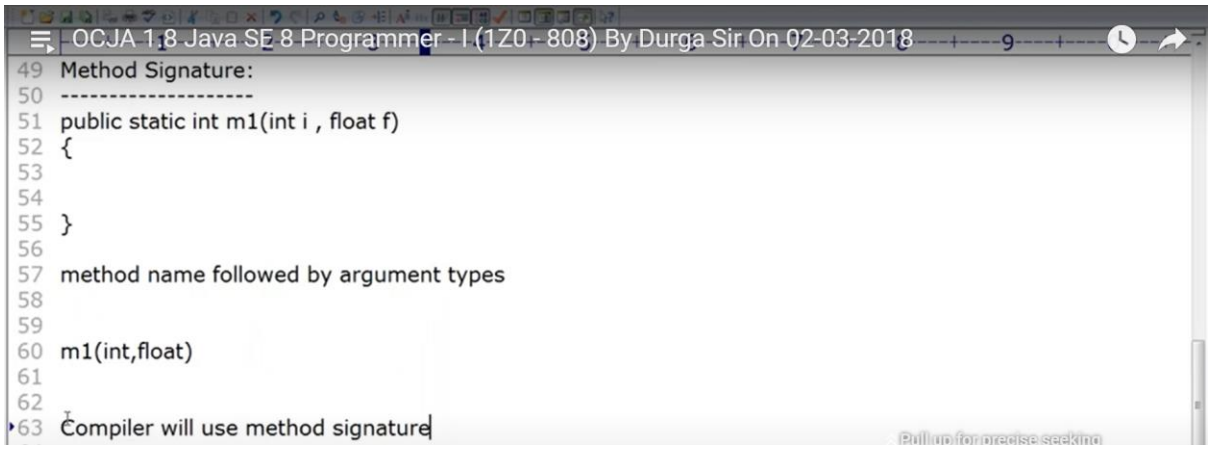
Composition

```
final class Car {

  private final Engine engine;

  Car(EngineSpecs specs) {
    engine = new Engine(specs);
  }

  void move() {
    engine.work();
  }
}
```

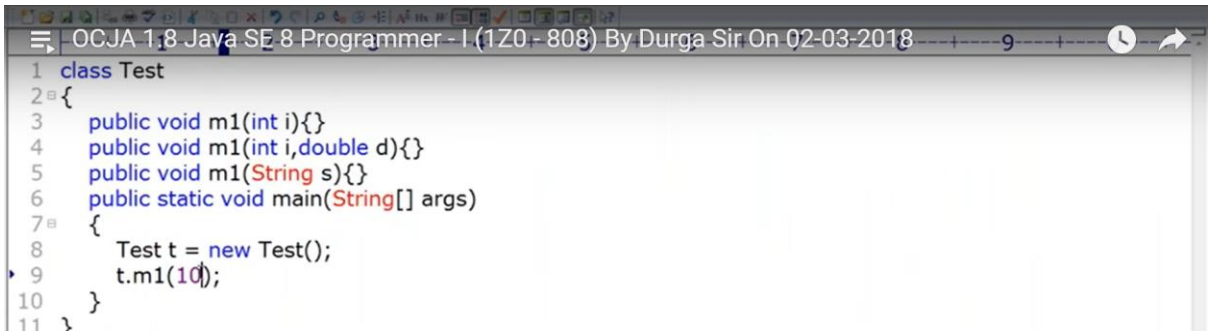
Aggregation

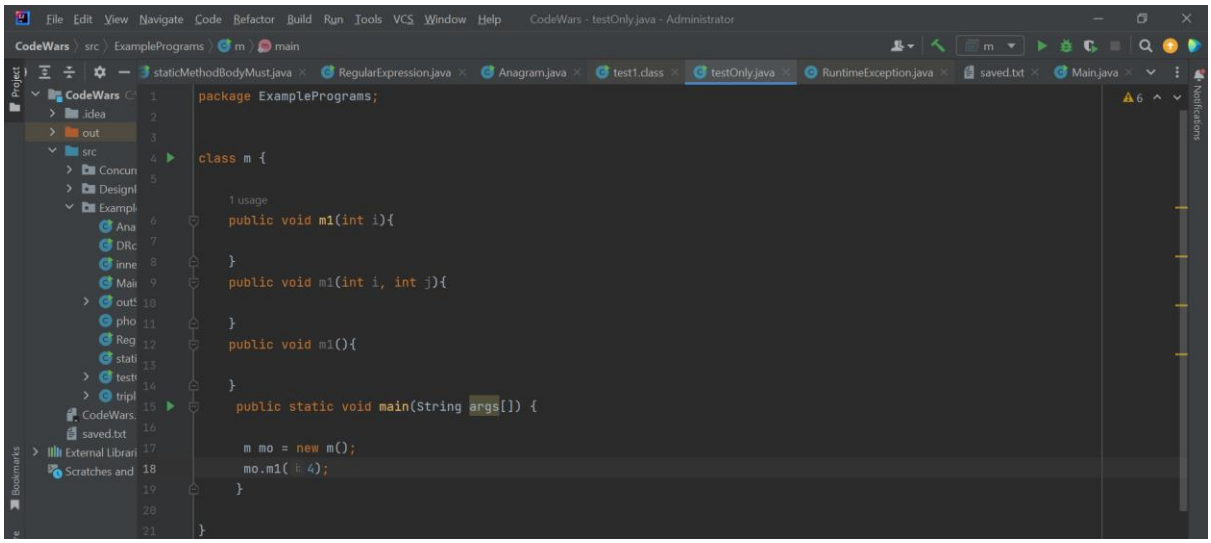
```
final class Car {

  private Engine engine;

  void setEngine(Engine engine) {
    this.engine = engine;
  }

  void move() {
    if (engine != null)
      engine.work();
  }
}
```

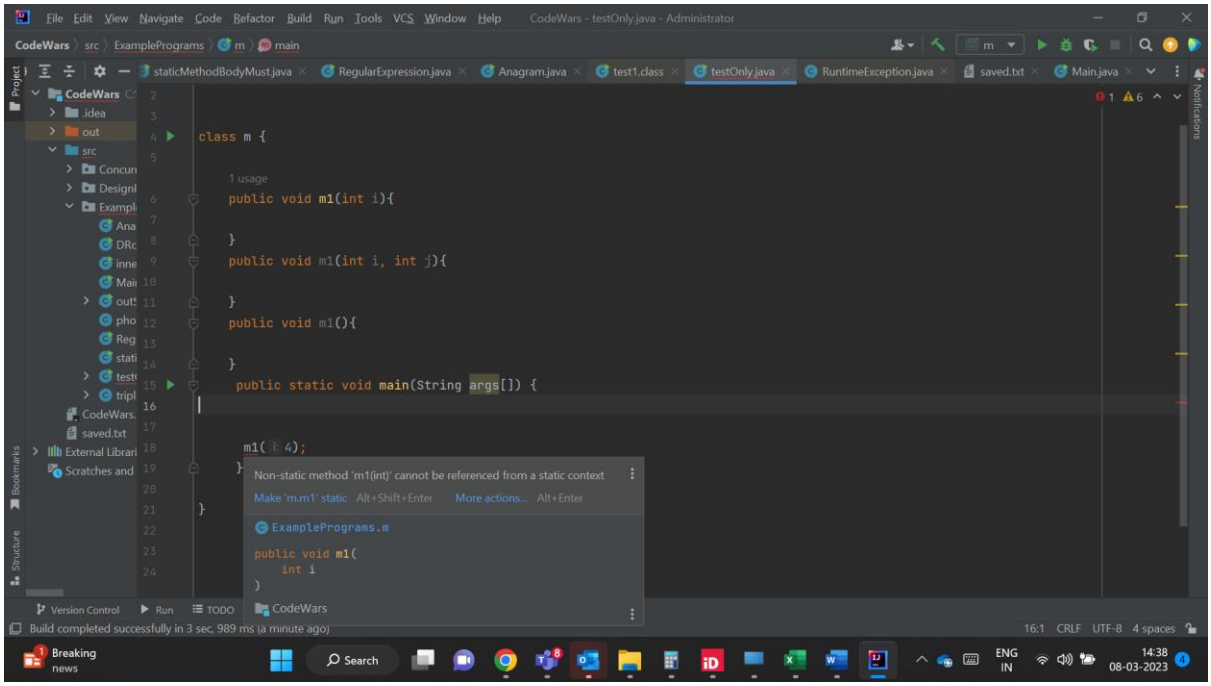
In the case of composition, the Engine is completely encapsulated by the Car. There is no way for the outside world to get a reference to the Engine. The Engine lives and dies with the car. With aggregation, the Car also performs its functions through an Engine, but the Engine is not always an internal part of the Car. Engines may be swapped, or even completely removed. Not only that, but the outside world can still have a reference to the Engine, and tinker with it regardless of whether it's in the Car.



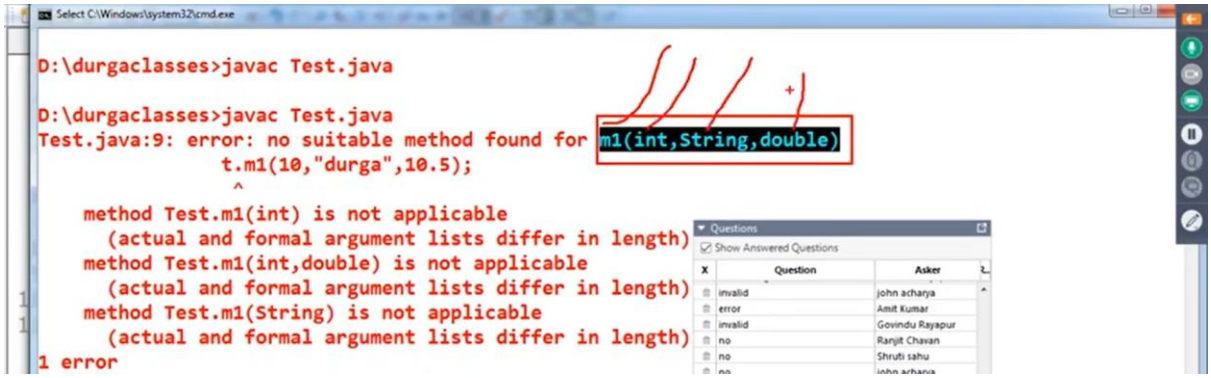
Look below the methods are not static but we are able to access non static method from main method by object creation



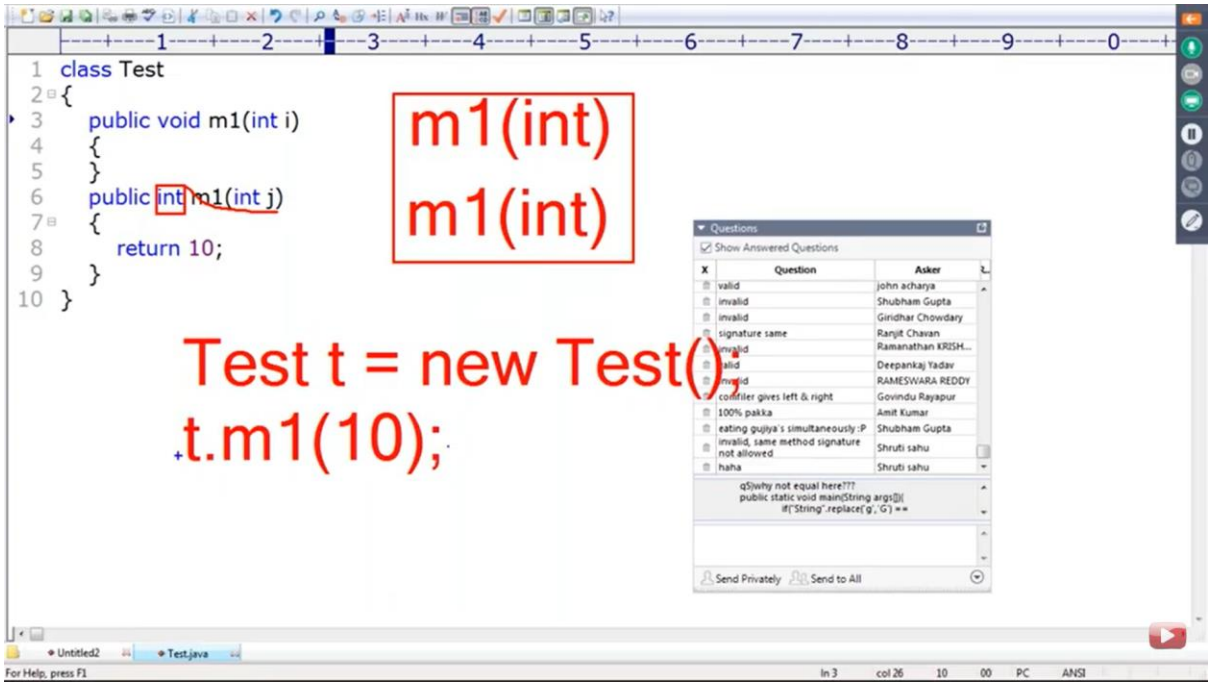




Compiler will always check method signature when solving methods



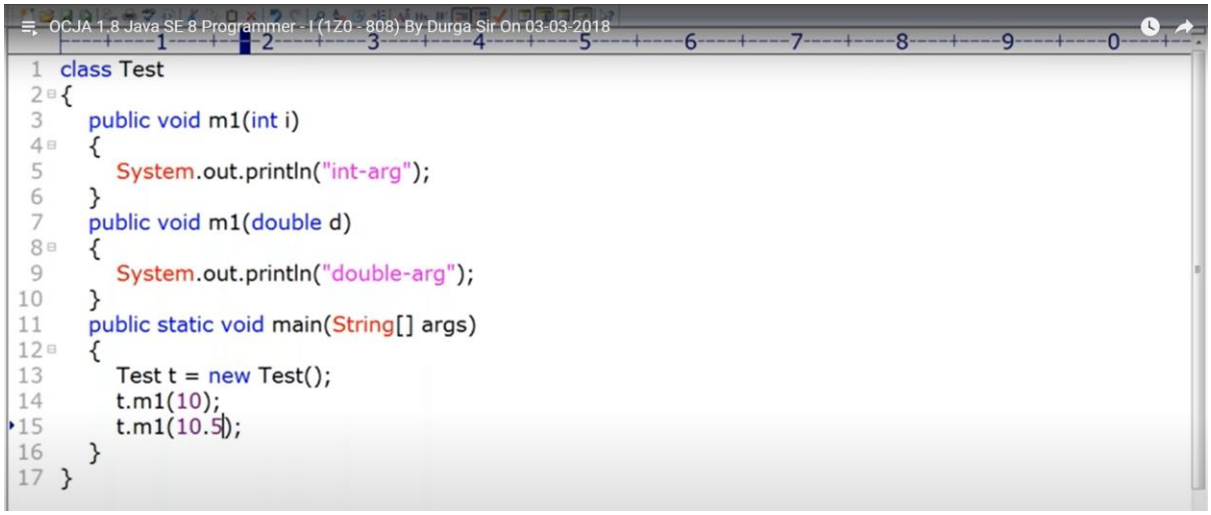
Two methods with same signature and same return type is not possible



Overloading

Method with diff arg types , different sequence and no of args. Even if they are present in child class if the sig are diff its called overloading

```
11
12 Java:
13 abs(int)
14 abs(long)
15 abs(float)
16 abs(dbouble)
17
18
19 m1(int i,float f)
20 m1(float f,int i)
```



In Overloading the method resolution taken care by compiler, which method to be executed, based **on ref type**. Hence its resolved during compile time its called compile time polymorphism.

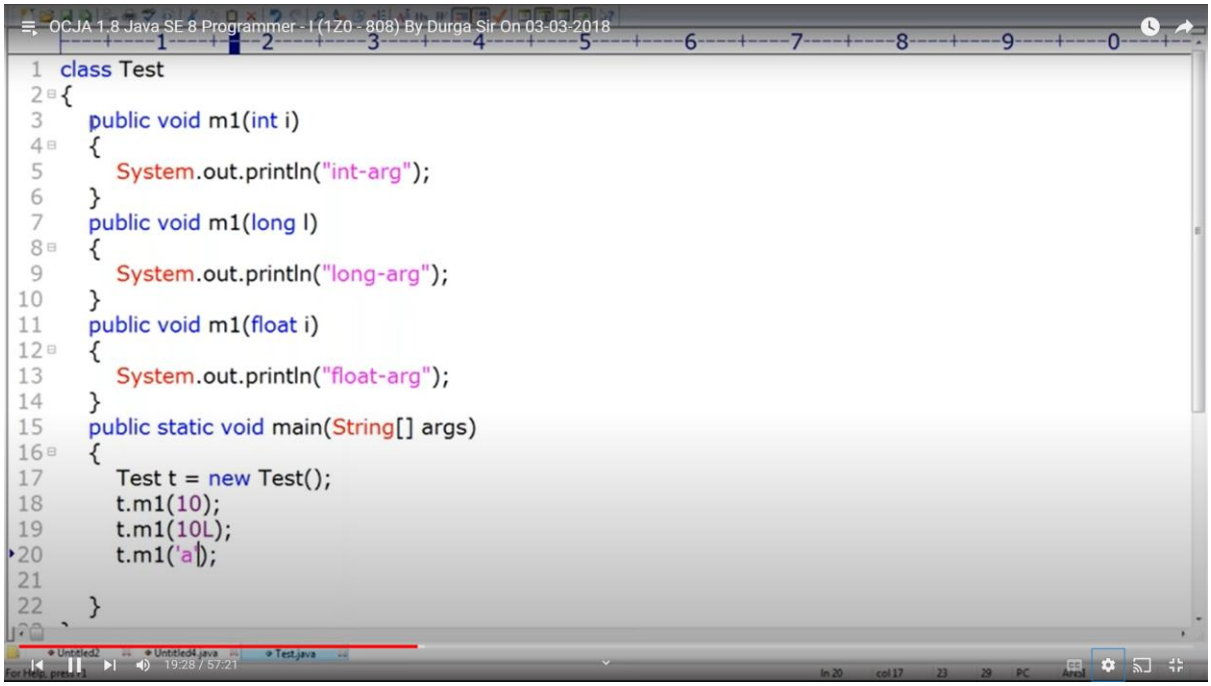
Ref type – Test T. Early binding since its resolved compile time

```
1  class Test
2  {
3     public void m1(int i)
4     {
5        System.out.println("int-arg");
6     }
7     public void m1(double d)
8     {
9        System.out.println("double-arg");
10    }
11    public static void main(String[] args)
12    {
13       Test t = new Test();
14       t.m1(10);
15       t.m1(10.5);
16    }
17 }
```

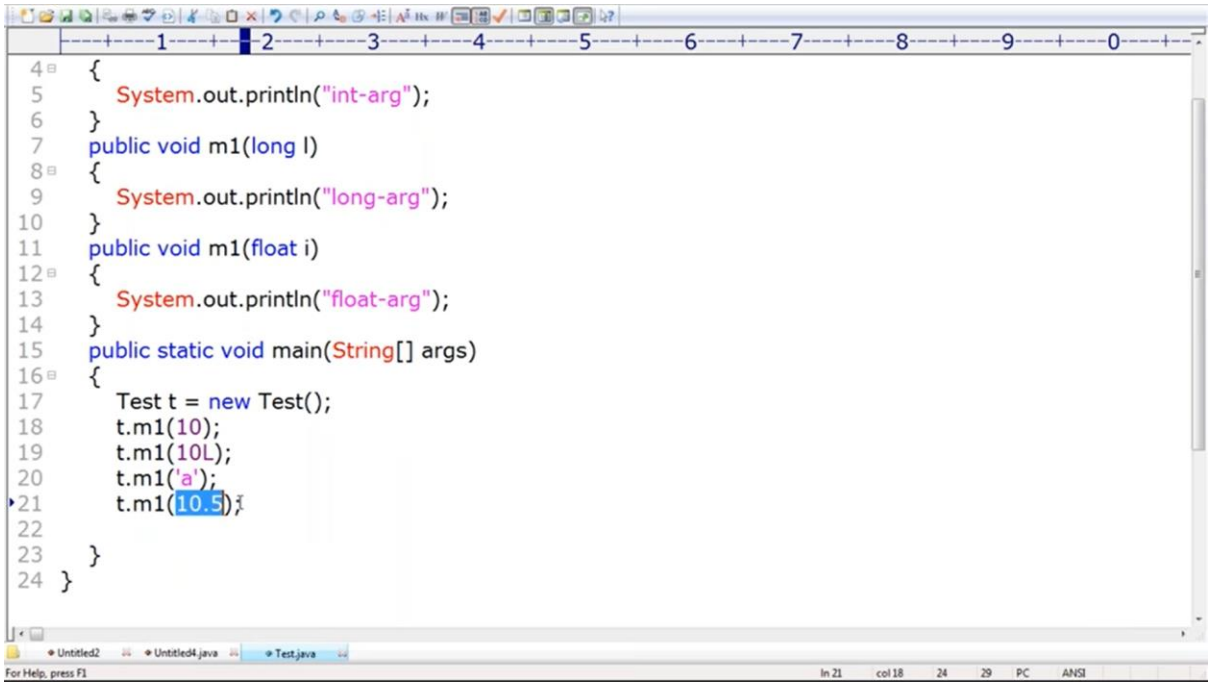
In overloading automatic promotion is taken care. If exact match is not found promote to next level. If int is not available then call long

Byte->Short-> int->Long

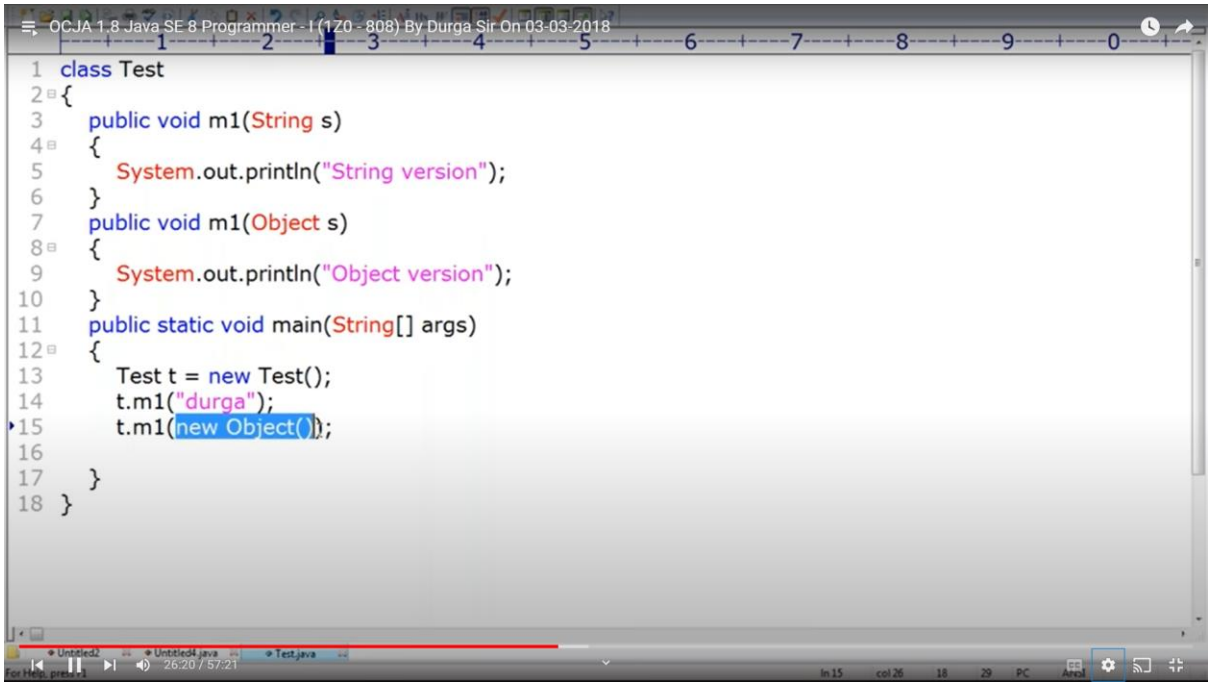
Char-> int



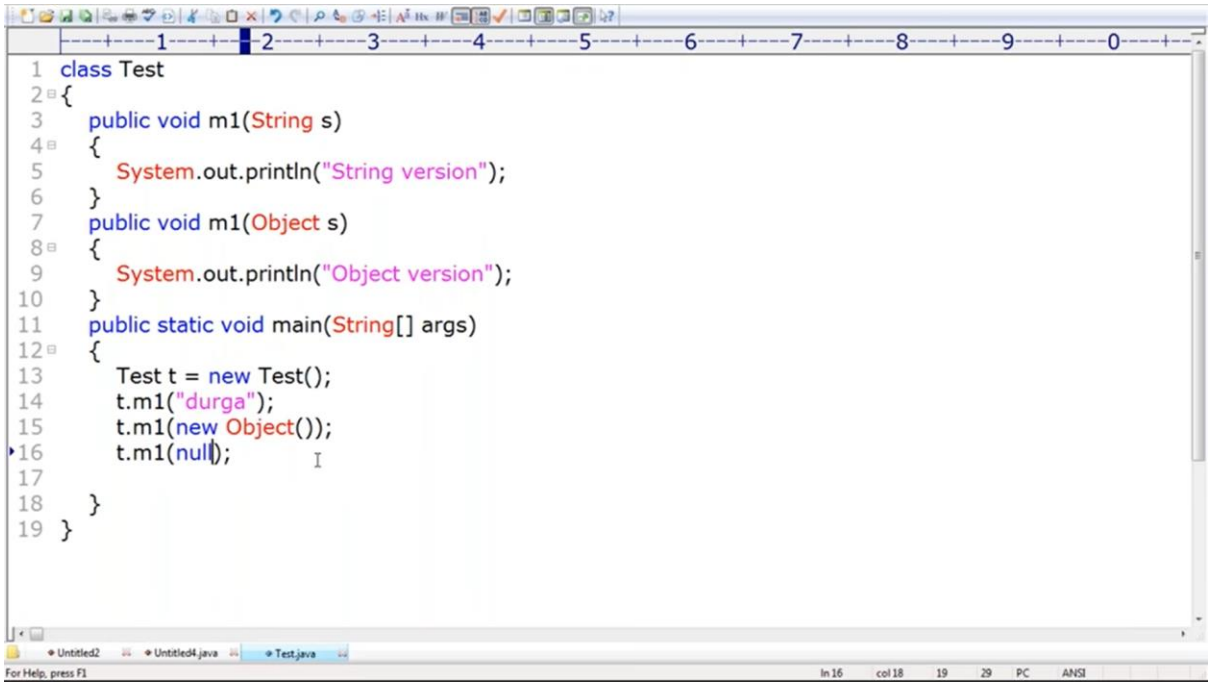
Double to float not possible below compile time error



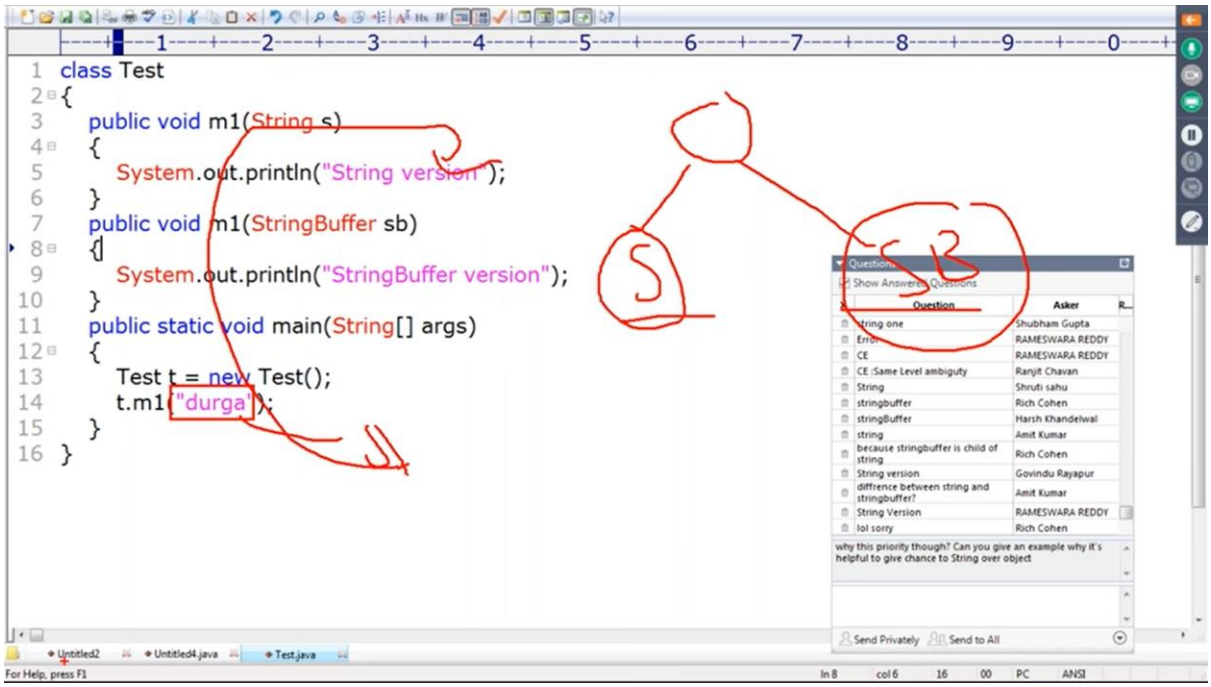
Below is correct

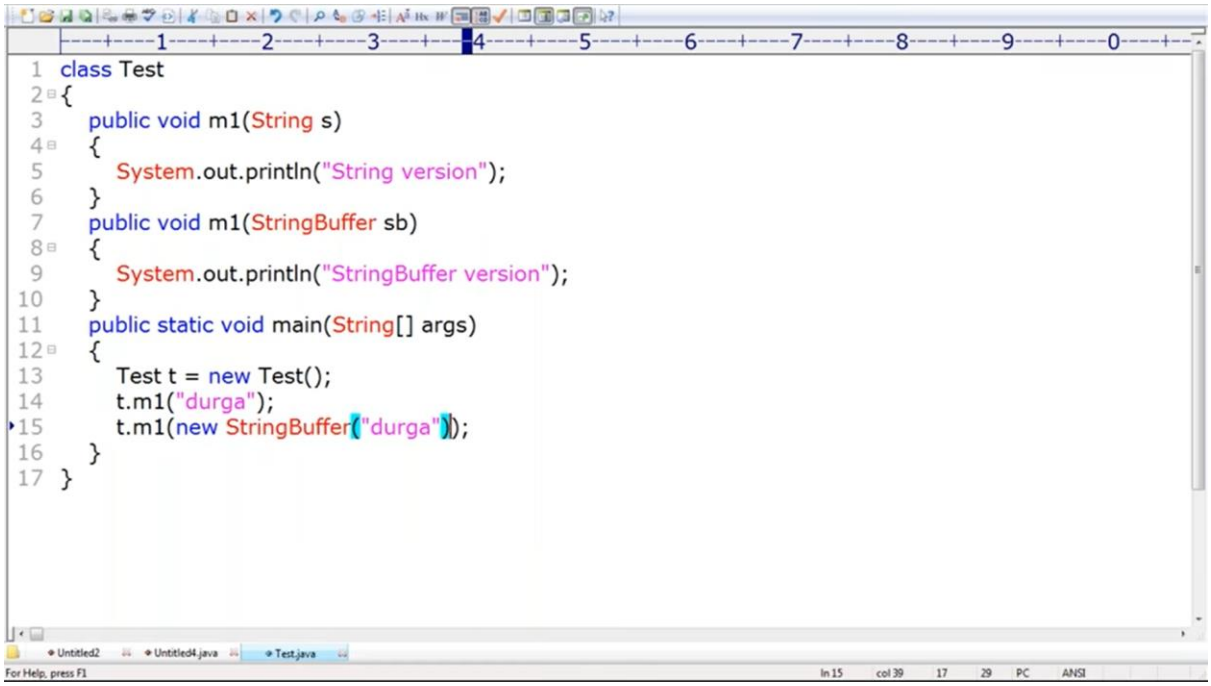


Null is valid object for String and object, but in the case low level or child argument -> String is going to get pref. More specific argument will get the highest priority. If no spec arg then high level arg

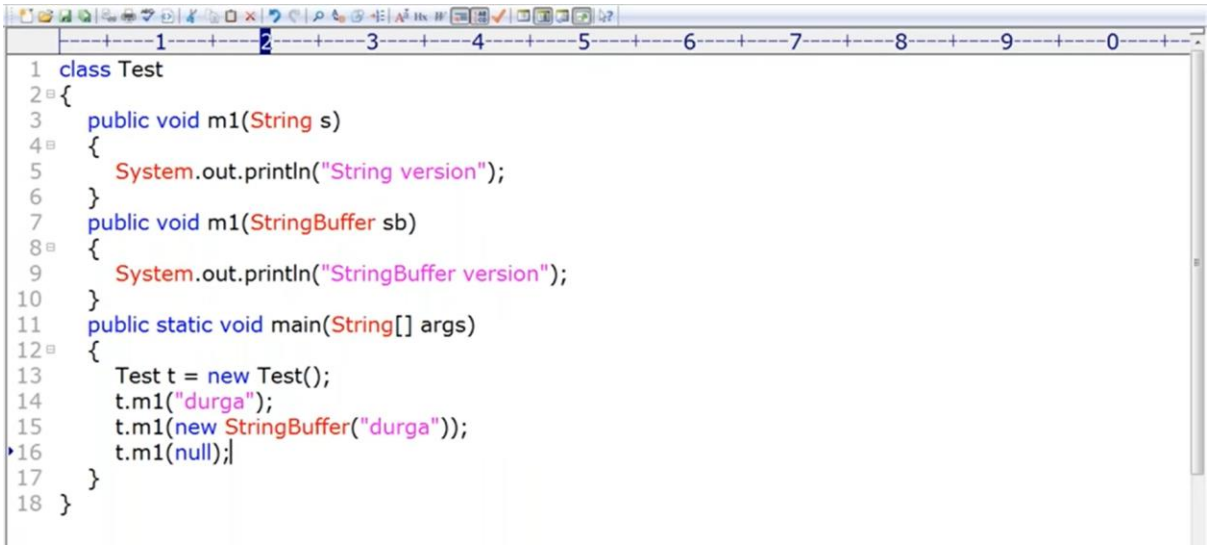


Exact match gets high priority

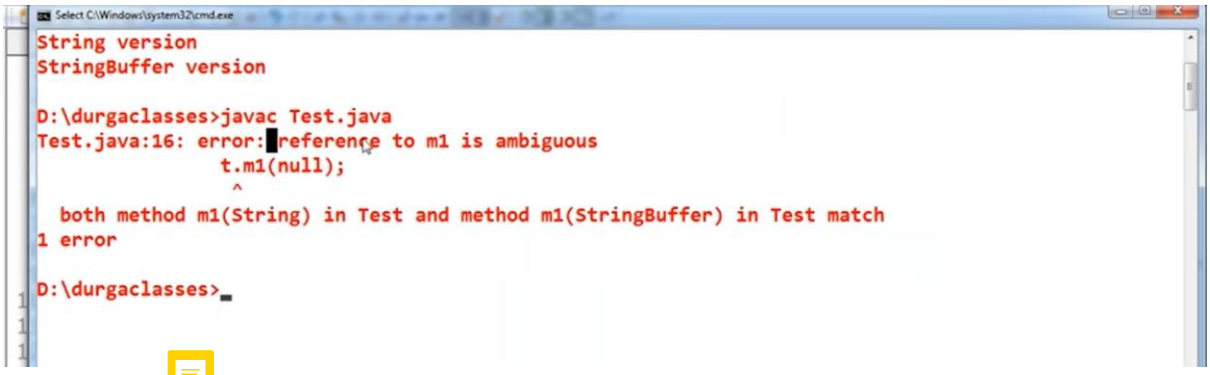




Compile time error, since both are specific

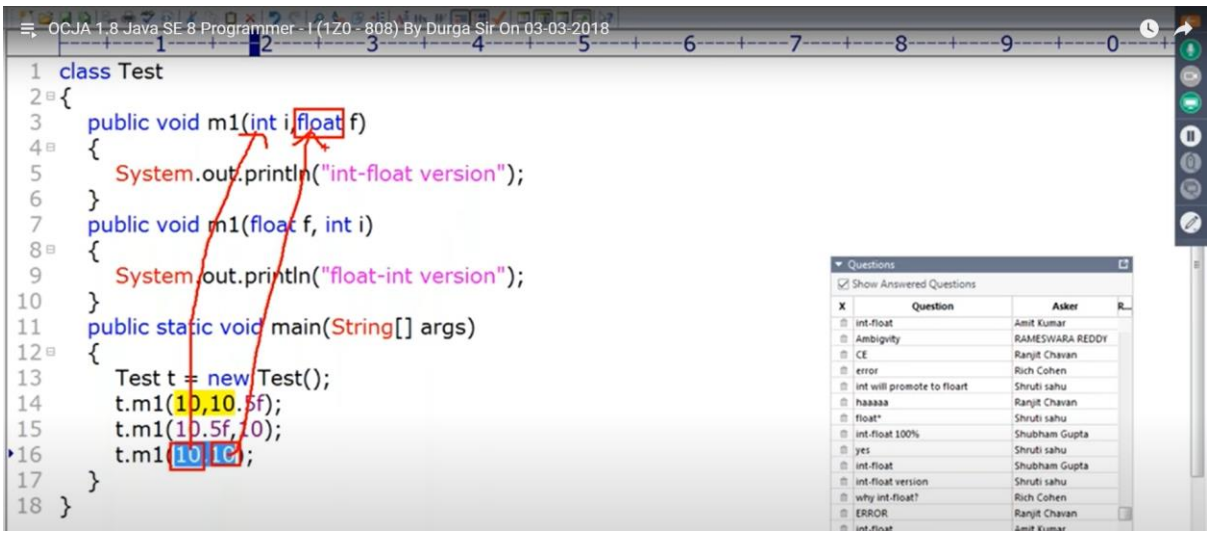


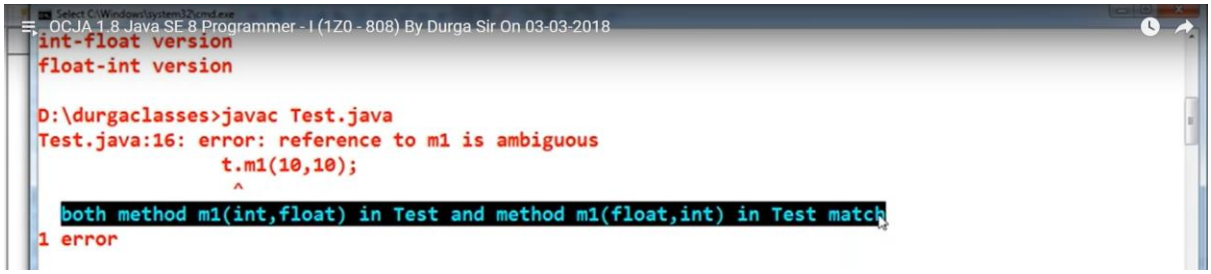
```
class Test
{
   public void m1(String s)
   {
      System.out.println("String version");
   }
   public void m1(StringBuffer sb)
   {
      System.out.println("StringBuffer version");
   }
   public static void main(String[] args)
   {
      Test t = new Test();
      t.m1("durga");
      t.m1(new StringBuffer("durga"));
      t.m1(null);
   }
}
```



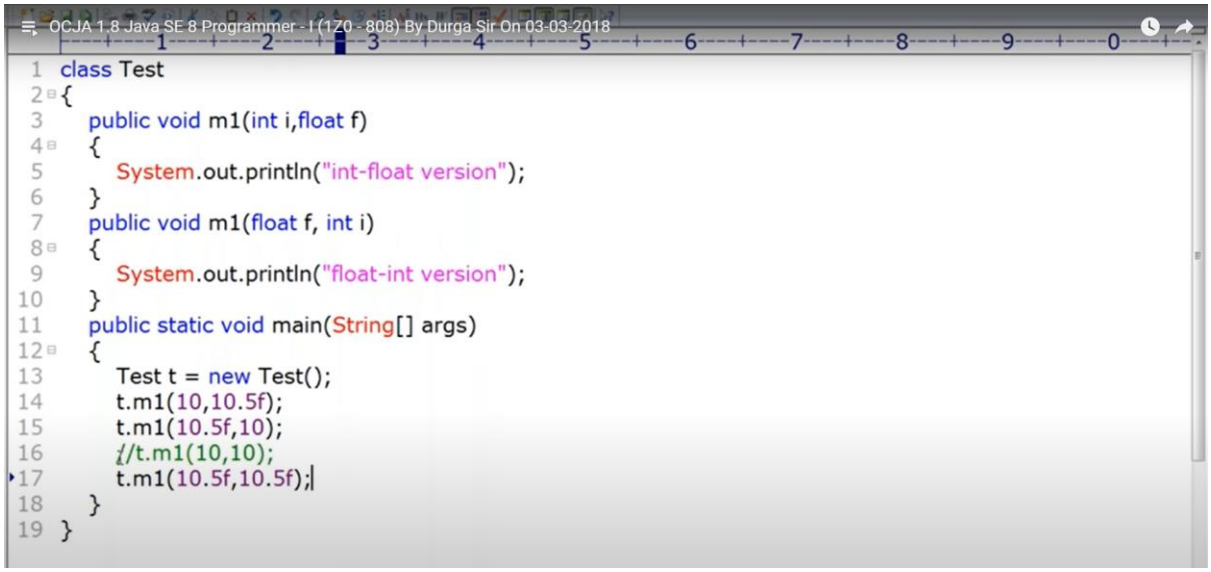
```
String version
StringBuffer version

D:\durgaclasses>javac Test.java
Test.java:16: error: reference to m1 is ambiguous
                t.m1(null);
                ^
  both method m1(String) in Test and method m1(StringBuffer) in Test match
1 error

D:\durgaclasses>
```

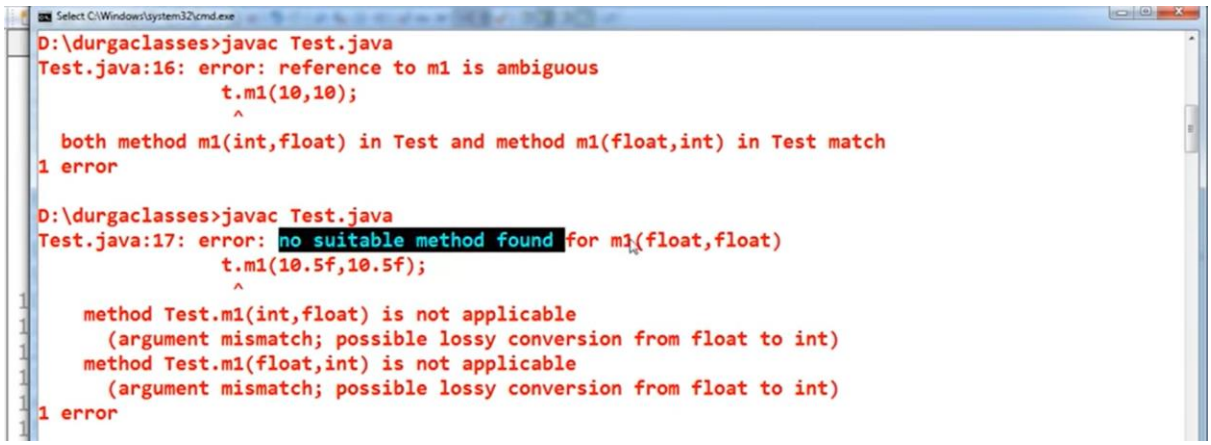
Int to float promotion is possible, but why are u seeing int first see second method go from right to left and compare args. There is not left to right or right to left

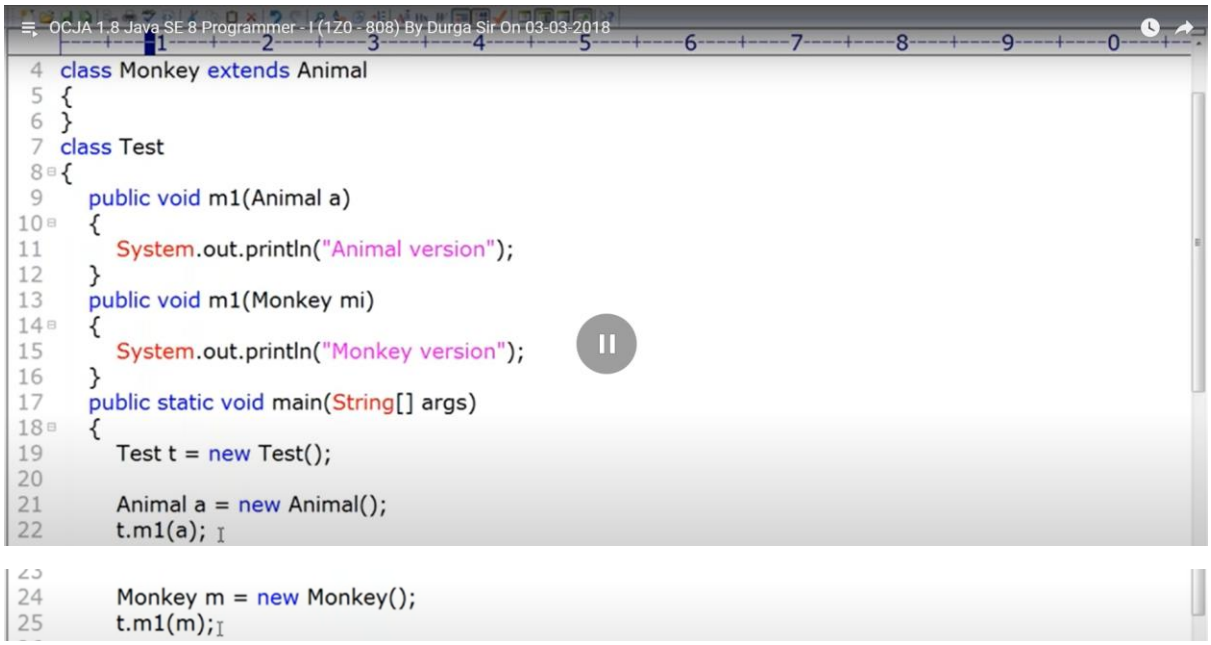




CE, cant promote from float to int.





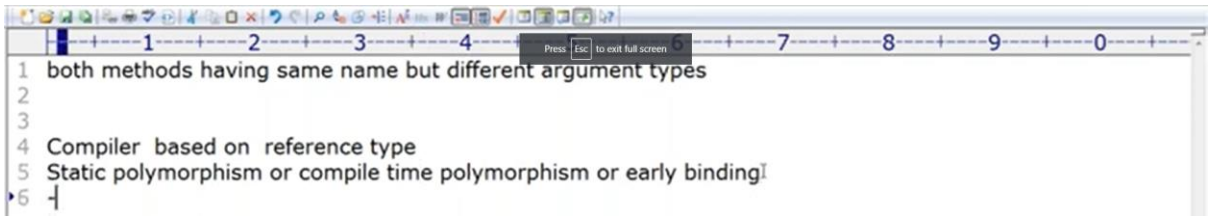


Compiler never checks about run time object it only checks the ref object. Here it checks m1(Animal)

It never checks run time object during overloading

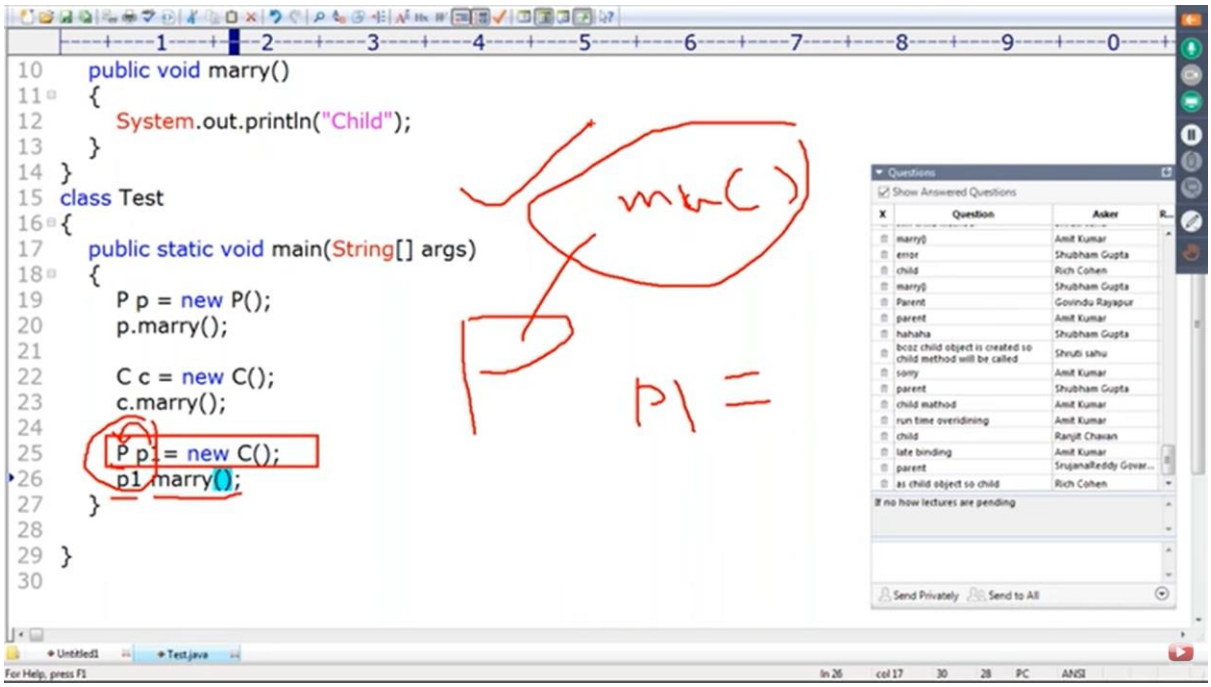
In Method overloading method resolution is taken care by compiler – Overriding by JVM

Overloading we consider only ref type – Overriding we consider Run time object

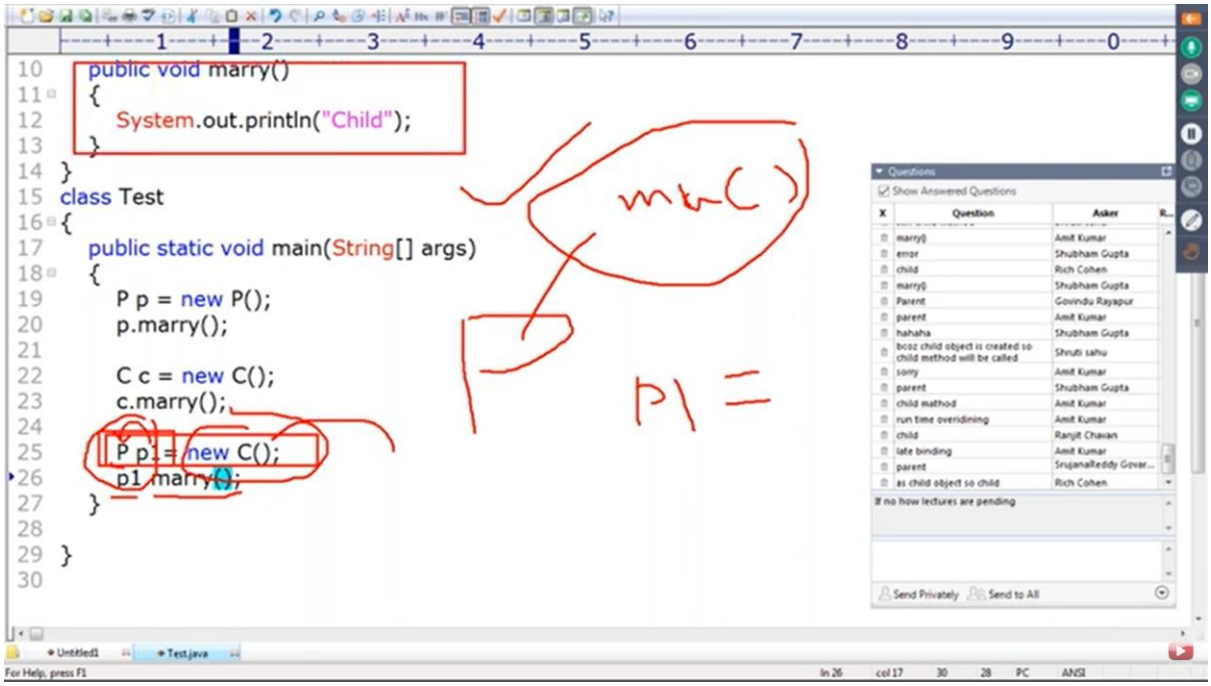


Overriding

Compiler job is to choose parent.



At run time it sees what object type it is, then sees if the method is there are not and choose child

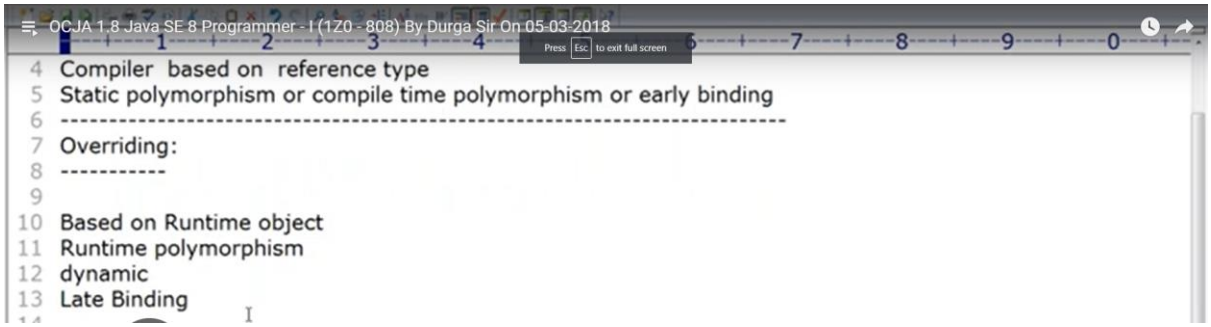


If marry method is not available in Parent class then we are going to get compile time error.

If marry method is not there

```
D:\durgaclasses>javac Test.java

D:\durgaclasses>javac Test.java
Test.java:23: error: cannot find symbol
                p1.marry();
                  ^
  symbol:   method marry()
  location: variable p1 of type P
1 error
```

Compiler is gonna check at compile time if marry method is there are not in Parent Class if not it will throw CE

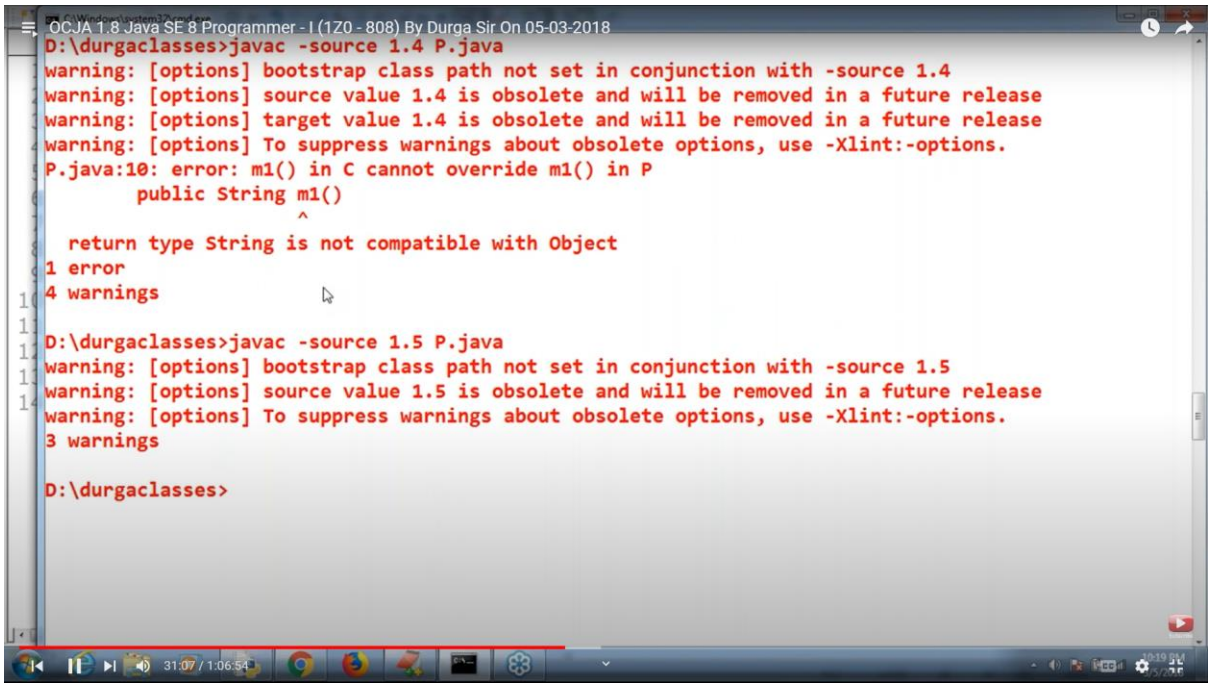


Order of signature is also very important

```
2.
1.4 Version only
1.5 version onwards
co-varient return types
```

Child method return type can be diff from Parent method return type- This is called covariant return type

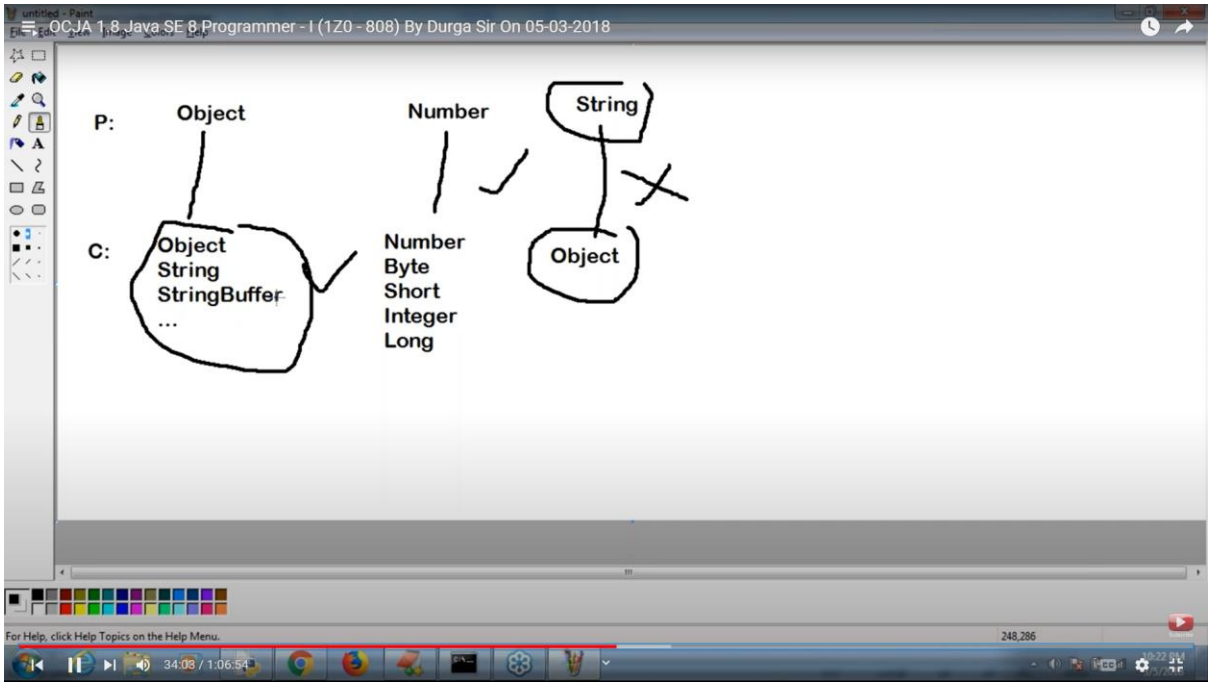
1.5 and 1.4



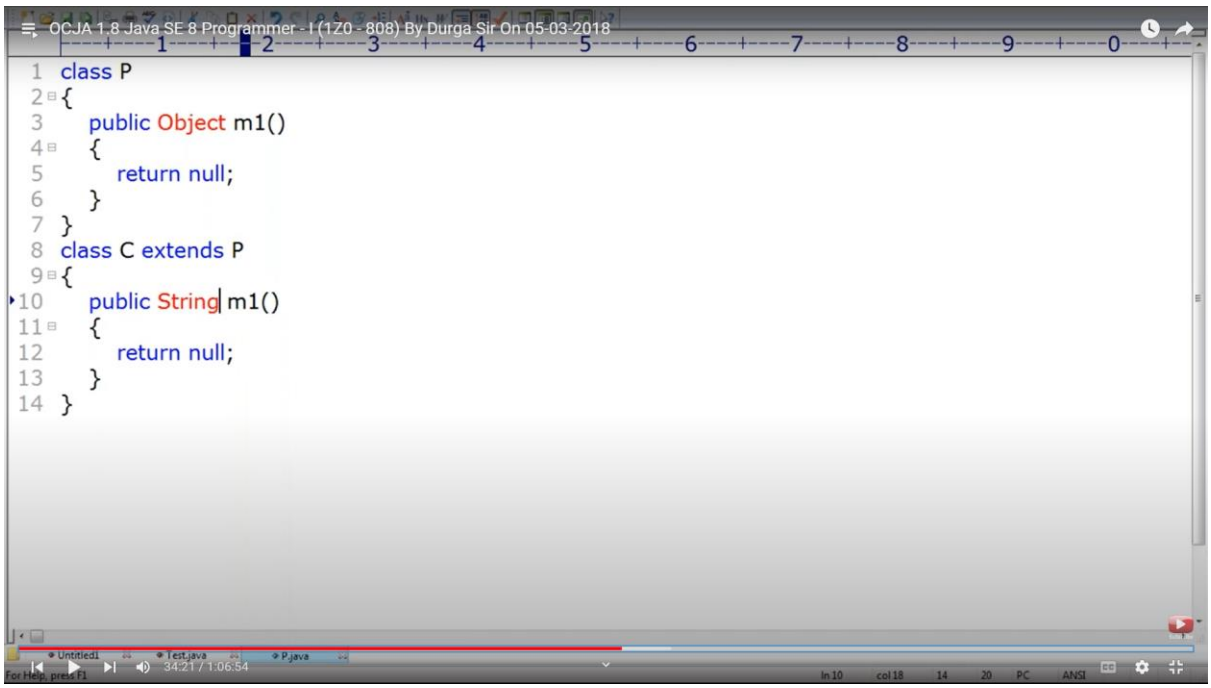
Return type criteria.

Child class method return type can be of same type a parent class method return type or child of parent return types

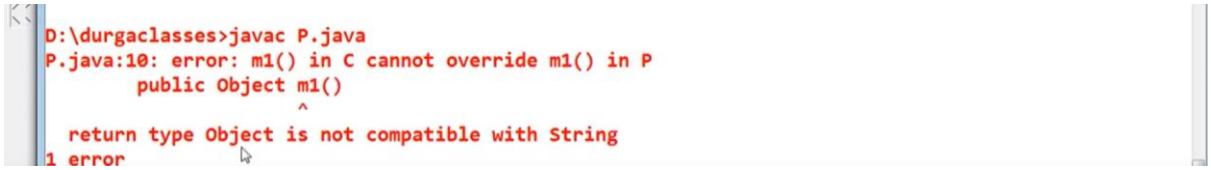
But parent method return type cannot be a child of child method return type



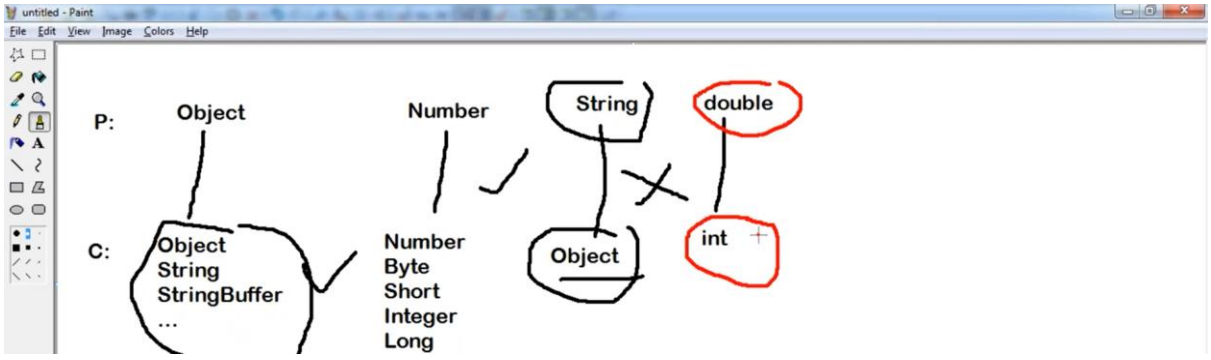
The below is correct



The below is an error



Covariant return type only matches for only for object types and not for primitive types

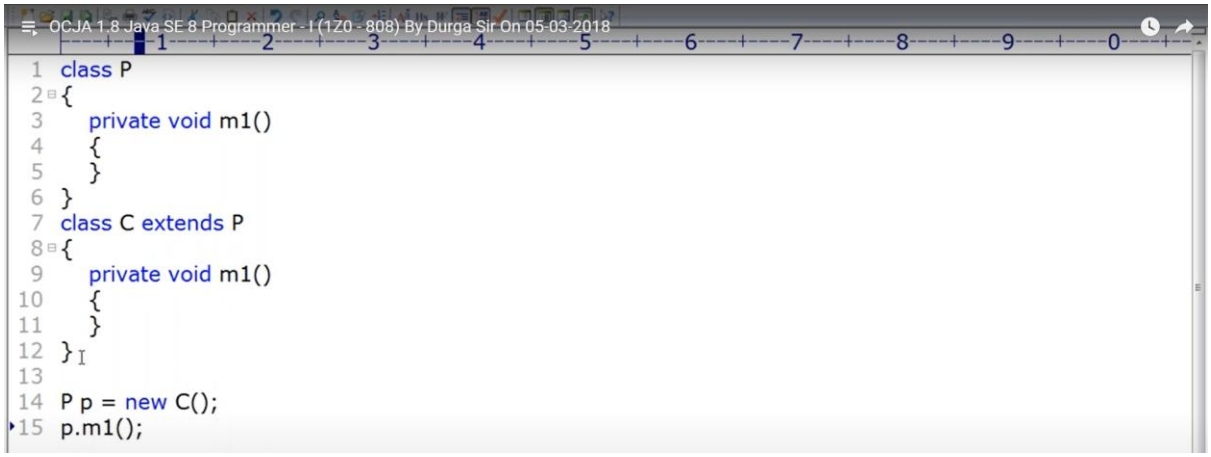


If parent is double child should also be double. Both should be int

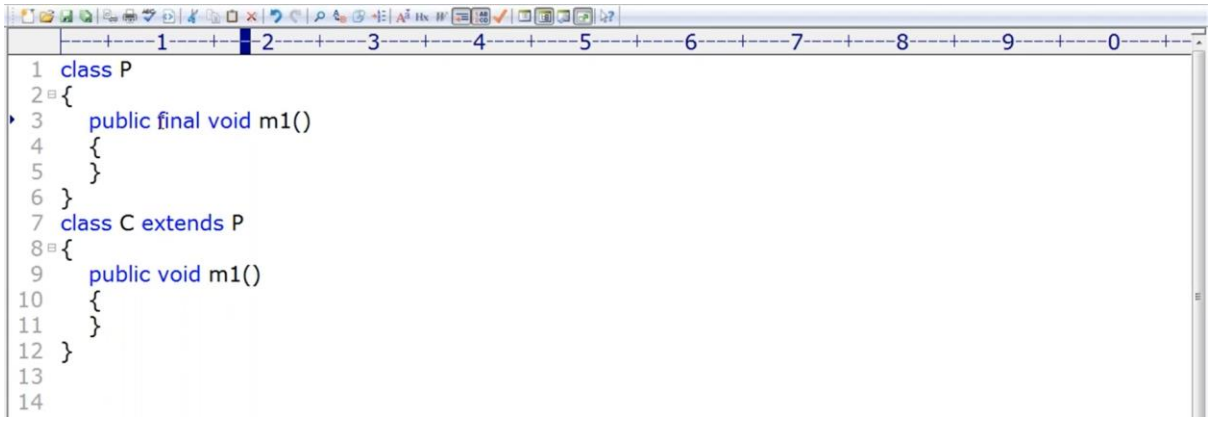
Private methods are not available for child class then overriding is not applicable

Same private methods can be defined in child class, but this is not overriding..

CE. P is private how can u call it from outside fool

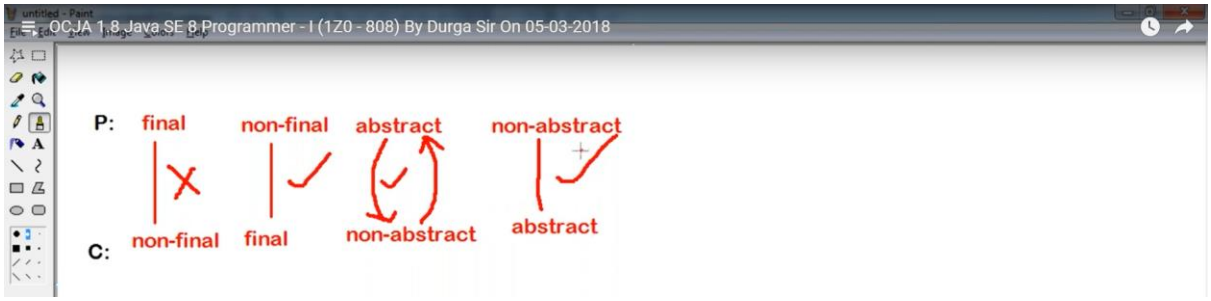


Overriding is not possible for final methods



Next level overriding is not possible

```
1  class P
2  {
3     public  void m1()
4     {
5     }
6  }
7  class C extends P
8  {
9     public final void m1()
10    {
11    }
12 }
13
```



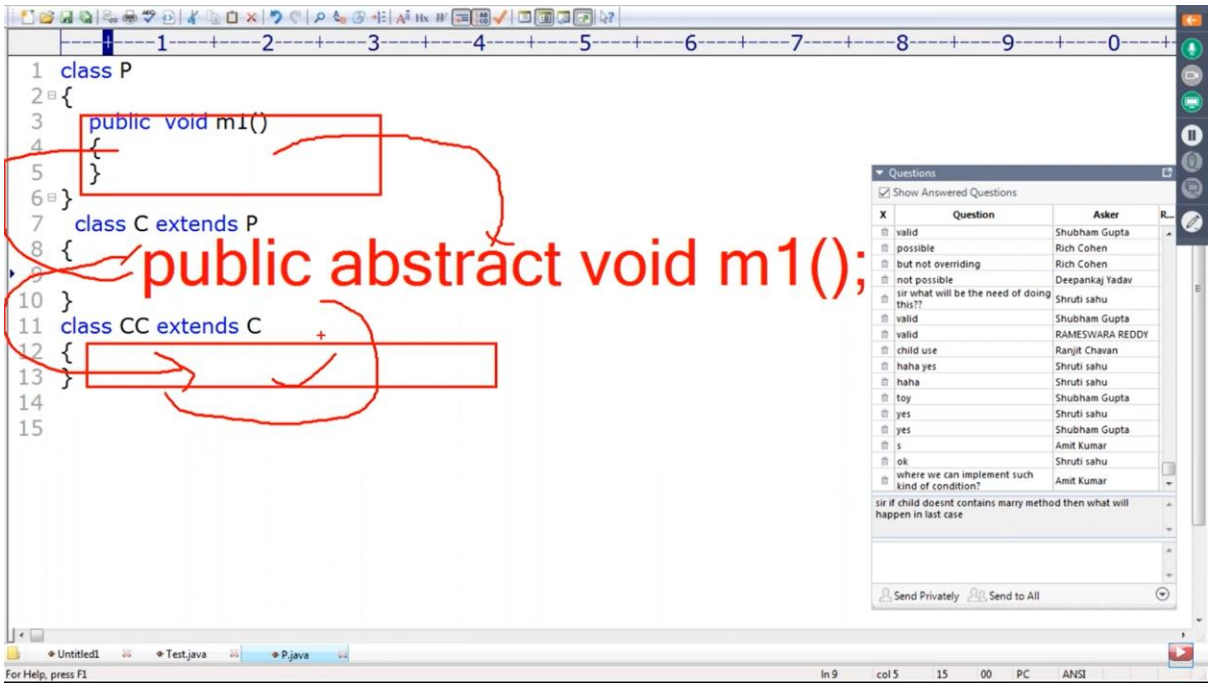

```
1  class P
2  {
3     public  void m1()
4     {
5     }
6  }
7  abstract class C extends P
8  {
9     public abstract void m1();
10 }
```

Child class extending c should implement. Abstract to non abstract and Non Abstract to Abstract is possible.

Why?

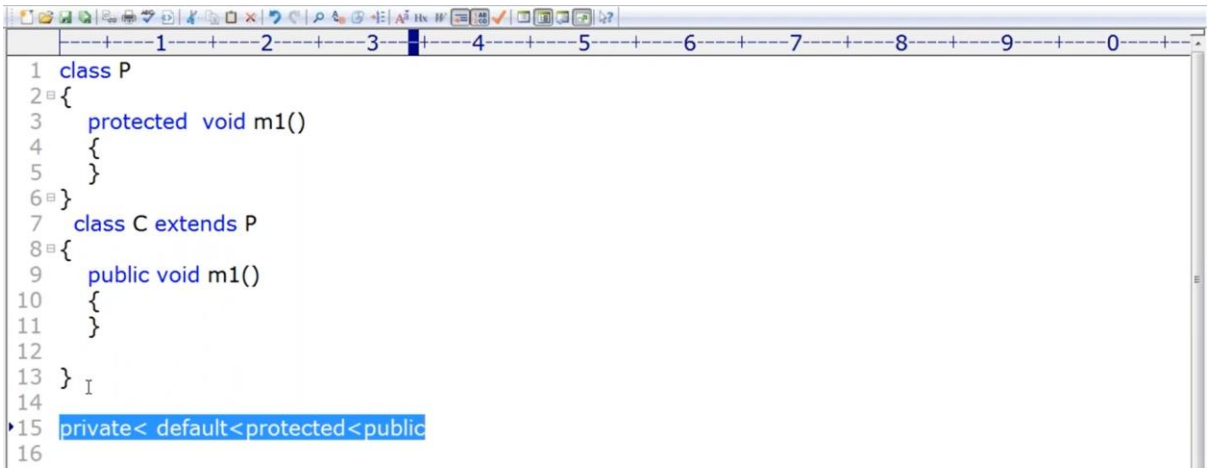
If I don't want to use parent method but I don't how to implement the method then we can use this way. Or we want child class to implement these methods

Parent method implementation is not available to next level child class and they should provide their own implementation



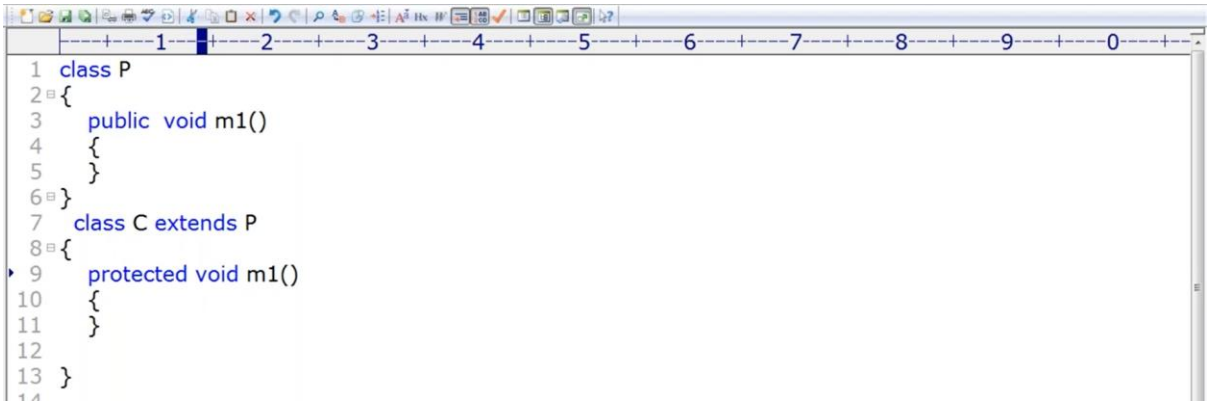
While overriding we cant reduce the scope of access modifiers , but u can increase the scope.

Protected to public is increasing the scope so its acceptable

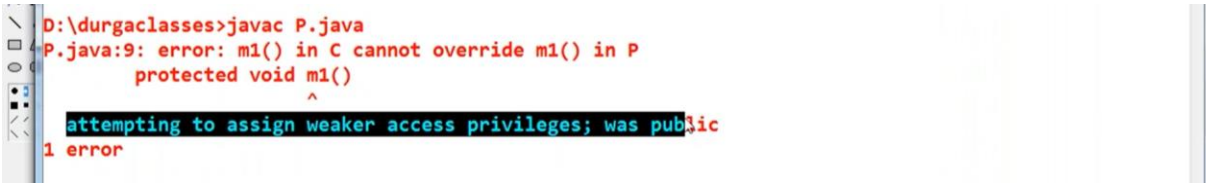


```
class P
{
   protected  void m1()
   {
   }
}
  class C extends P
{
   public void m1()
   {
   }

} 

private< default<protected<public
```

Below is invalid

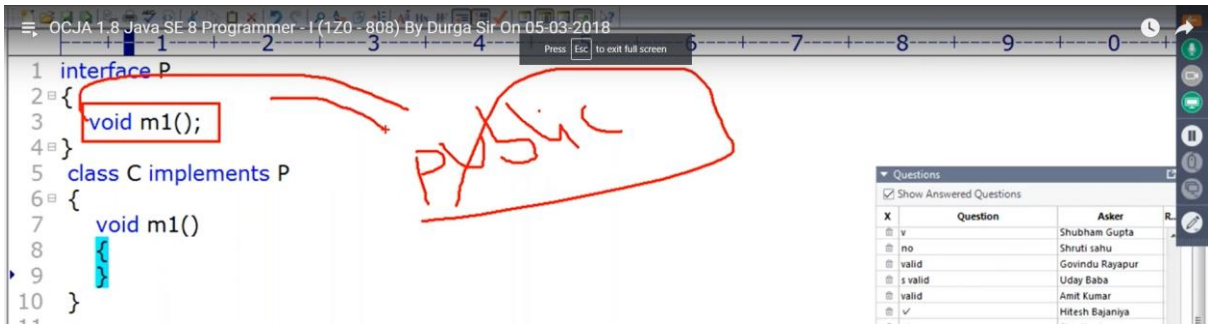


```
class P
{
   public  void m1()
   {
   }
}
  class C extends P
{
   protected void m1()
   {
   }

}
```



```
D:\durgaclasses>javac P.java
P.java:9: error: m1() in C cannot override m1() in P
       protected void m1()
                      ^
  attempting to assign weaker access privileges; was public
1 error
```

Every method in interface is by default public the child class method is default type which is lower scope . public method overriding should be Public only, cant reduce the scope of modifier



Same as above