Today Lecture 2 CS202

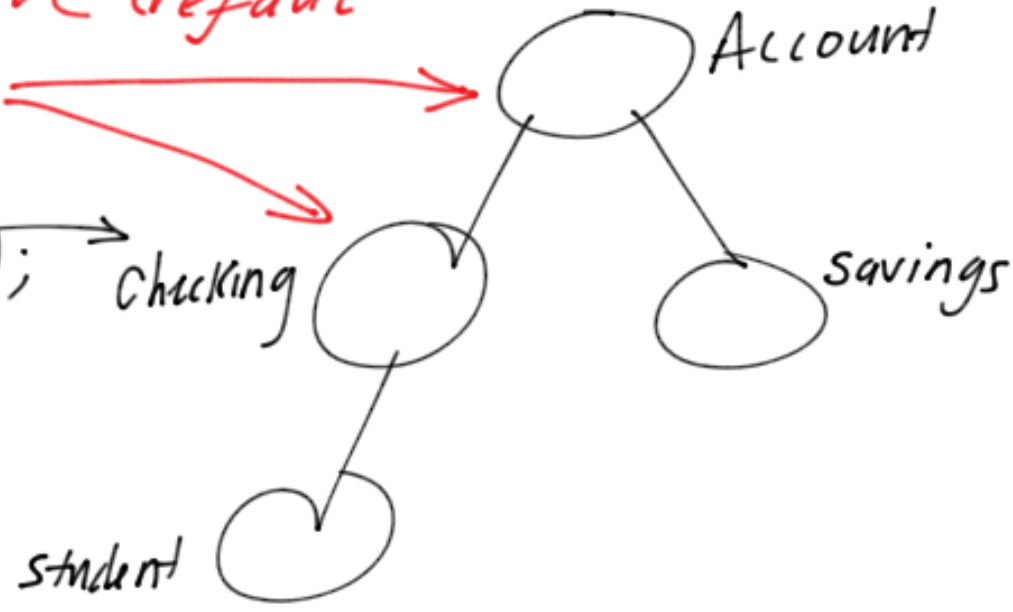
1) Topic #3 - Inheritance
   - single inheritance
   - multiple inheritance
2) Example Implementation
3) Preview of Dynamic Binding

plus ....
constructors
with arguments

Announcements

Default Constructors

Forgot to have defaut constructors




```
checking my_account;
checking array[15];
```

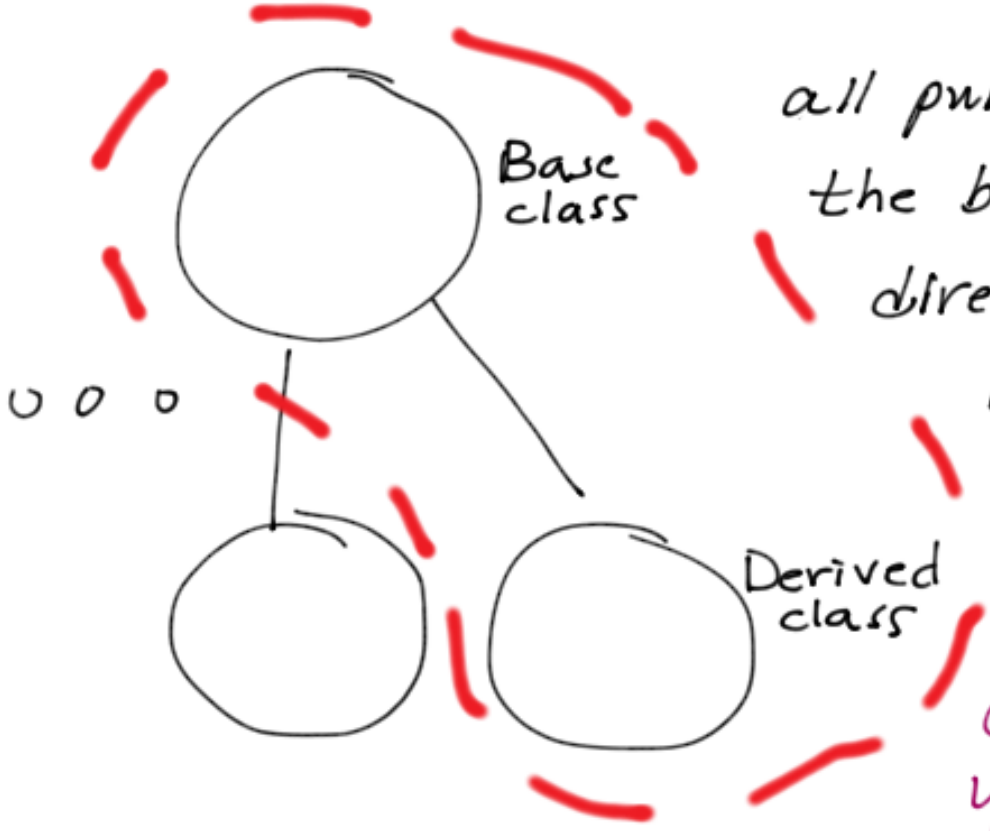
} won't compile

```
checking an_account(3.5);
```

} won't compile $\boxed{\text{If}}$ there isn't an initialization List

Inheritance - Single

Pubic Derivation:



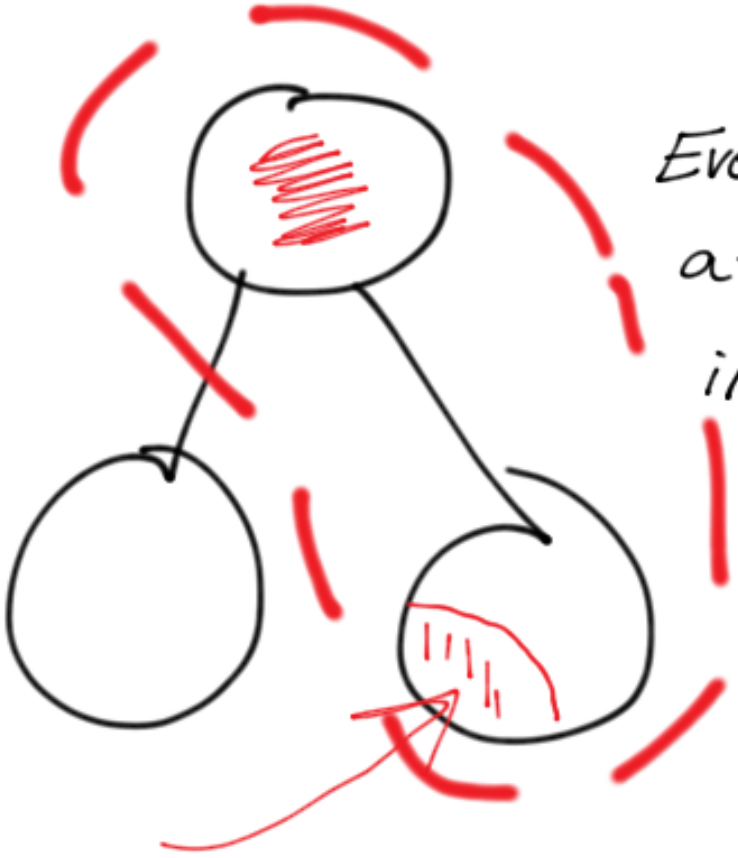
all public members in the base class are directly accessible by the "application" or client program

(any part of your code that creates an <u>object</u> of this class)

```
class derived : public base
{ public:

  protected:

  private:
};
```

this is what causes "public" derivation.

Protected Derivation

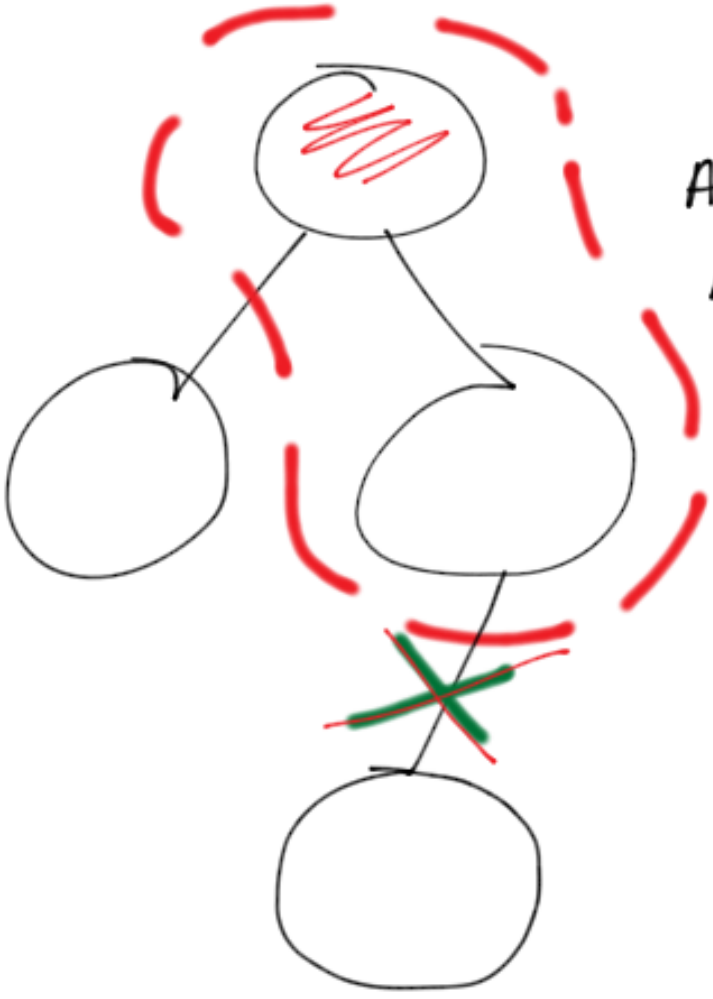


Everything Public acts as if it is protected in the derived class and it is not is available at all to the client/application program.

```
class derived:protected base
{
  public:

  protected:

  privated:

};
```

this allows us to essentially replace the public "client" interface of a class.

Private Derivation

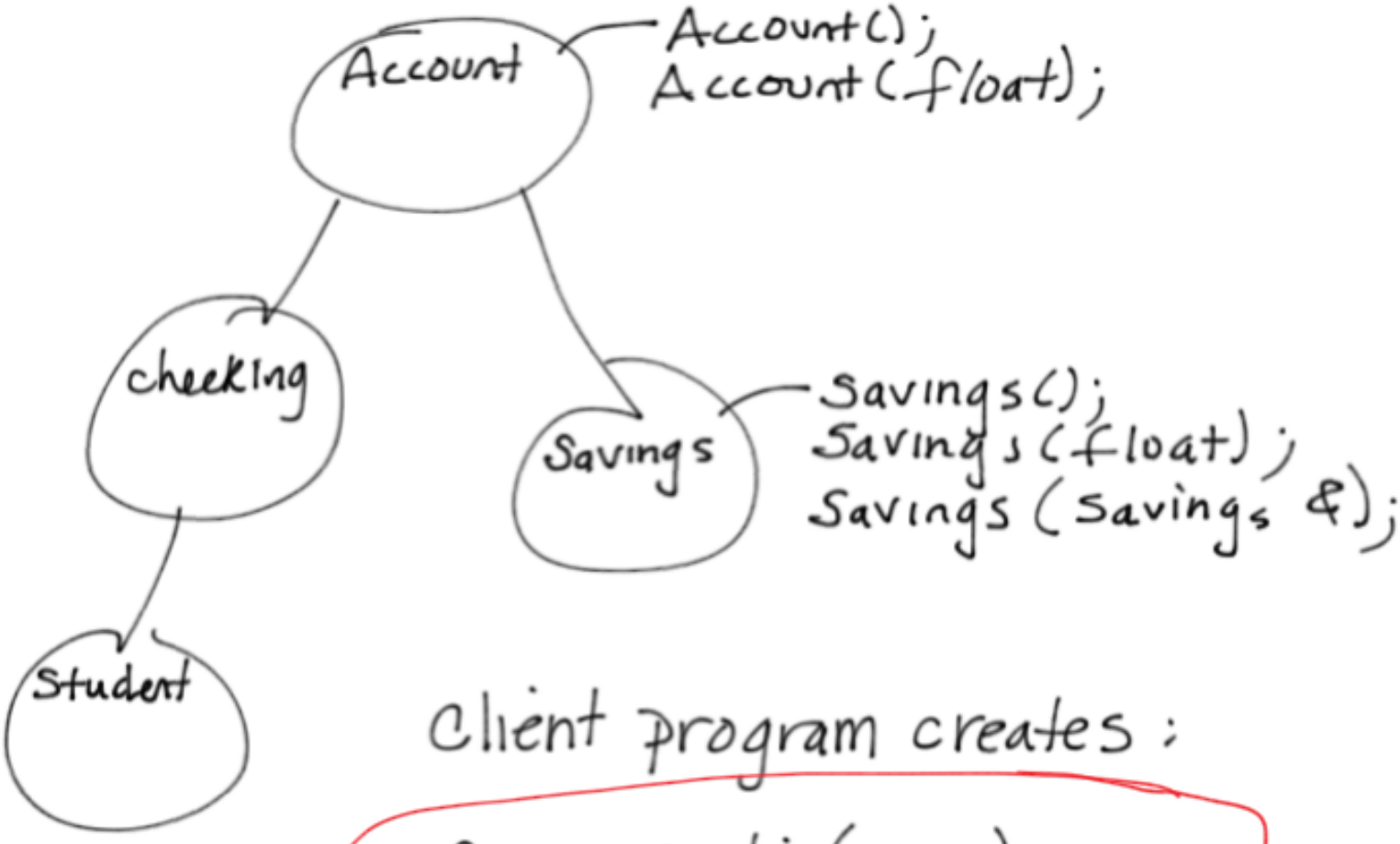


All public and protected members in the base class **are** accessible to the derived class **BUT** not to any subsequent derived classes NOR to the client/application

```
class derived : private base
{   public :
    protected :
    private :
};
```

(use this when you want to prohibit meaningful future derivation)

Constructors with Arguments



Client program creates:

Savings obj(100.0);

1. First the default constructor for the Account class is implicitly called.

2. Then, the savings constructor with the float argument is called

So, the information is not passed up to the parent.....

Initialization Lists

SOLUTION

In the implementation of the constructors we can add initialization lists.

They can be used to kick start the parents' constructor when arguments are involved

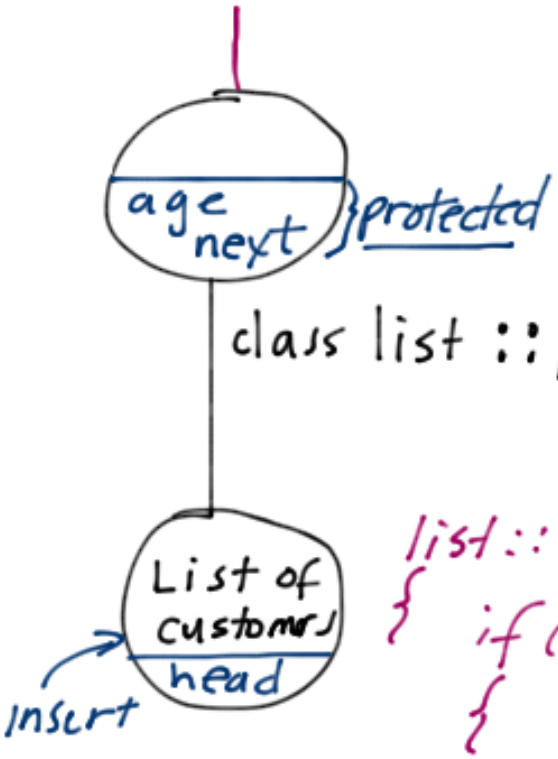
```
Savings::savings(float val) : Account(val), fee(9)
{
    ... // body of the function
}
```

initialization list

data member (Savings)

now the default constructor will not be involked when an object is created with a float passed as an argument

Watch out – what is wrong here:



class list :: public node ← watch out!

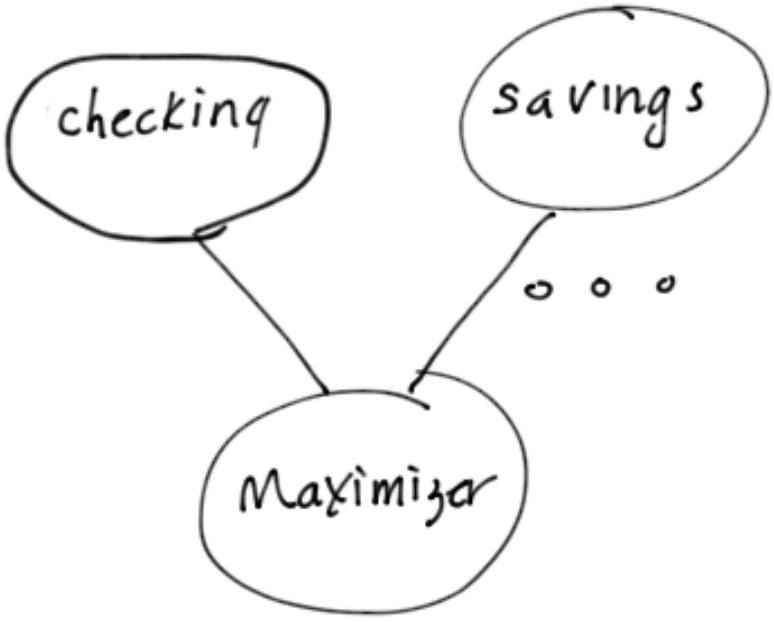
```
list::insert(const data & to_add)
{   if(!head)
    {   head = new node;
        head->next = NULL; //does not compile
        head->customerage = to_add.age;
```

also does not compile!

This hierarchy gives the derived class access to its base class' members (age and next) but above we made the mistake of being a $\boxed{\text{CLIENT}}$ of the base class. So, when using "head" in this case there is no hierarchy involved (and clients have no access to private or protected members)

Remember hierarchy is not for 1-to-many relationships

Multiple Inheritance

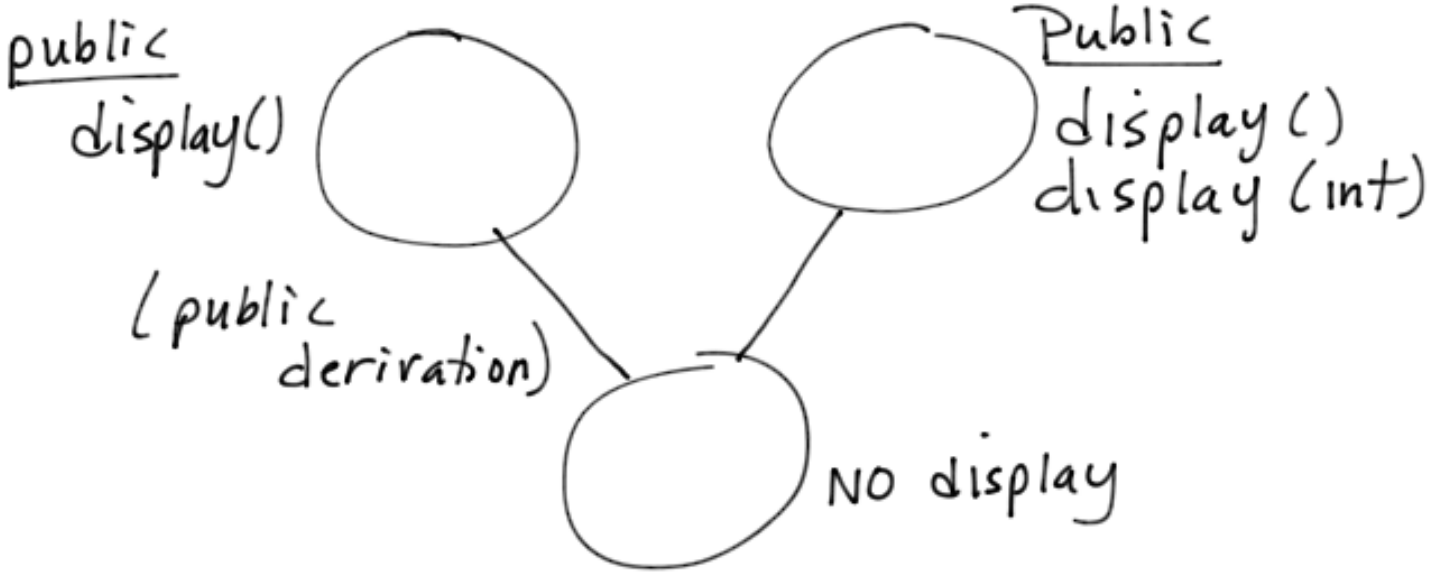



```
class maximizer: public checking, public savings
{
  :
```

comma separated list of base classes

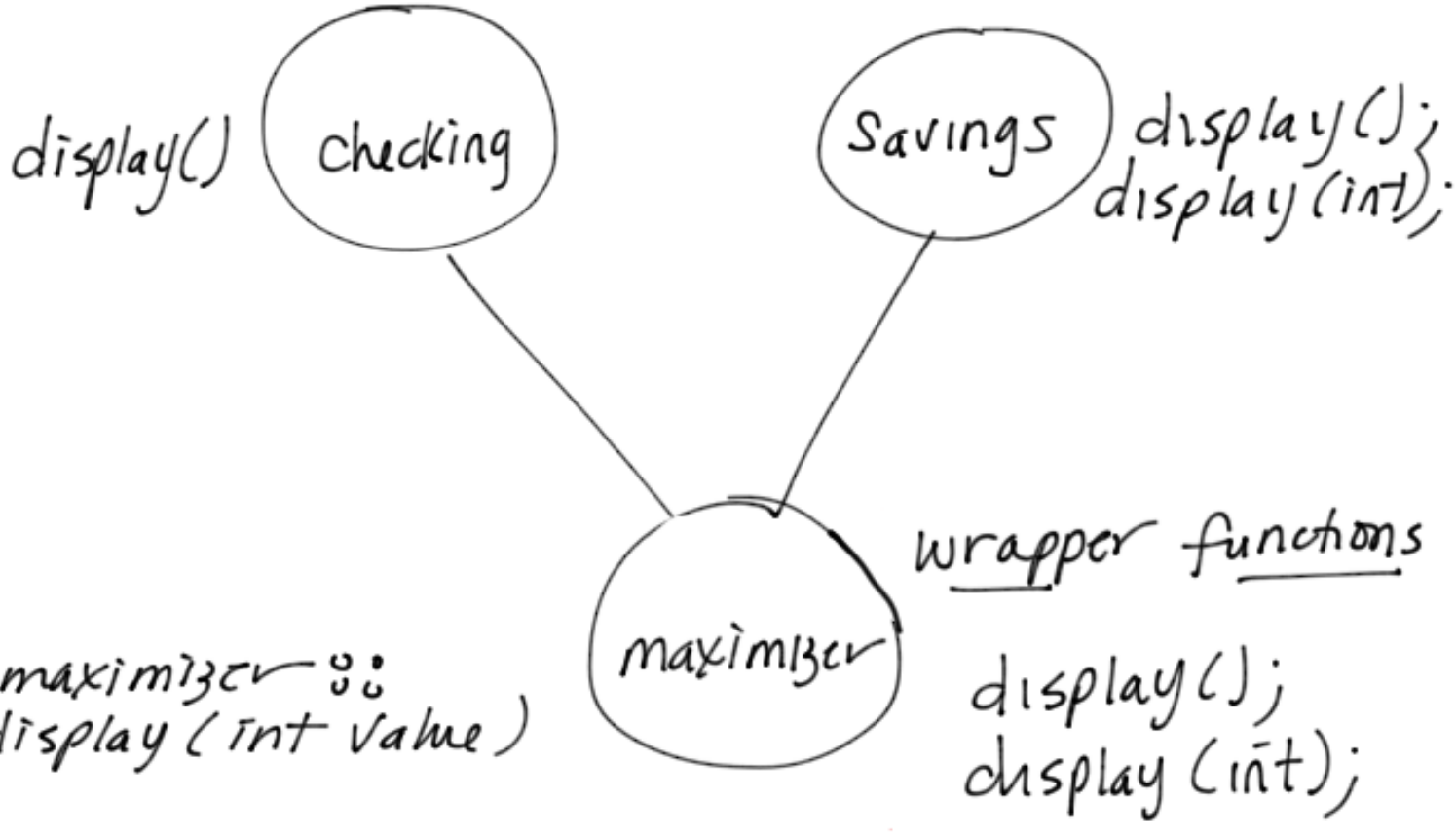
- derivation list specifies the order the constructors will be invoked

Same name members...

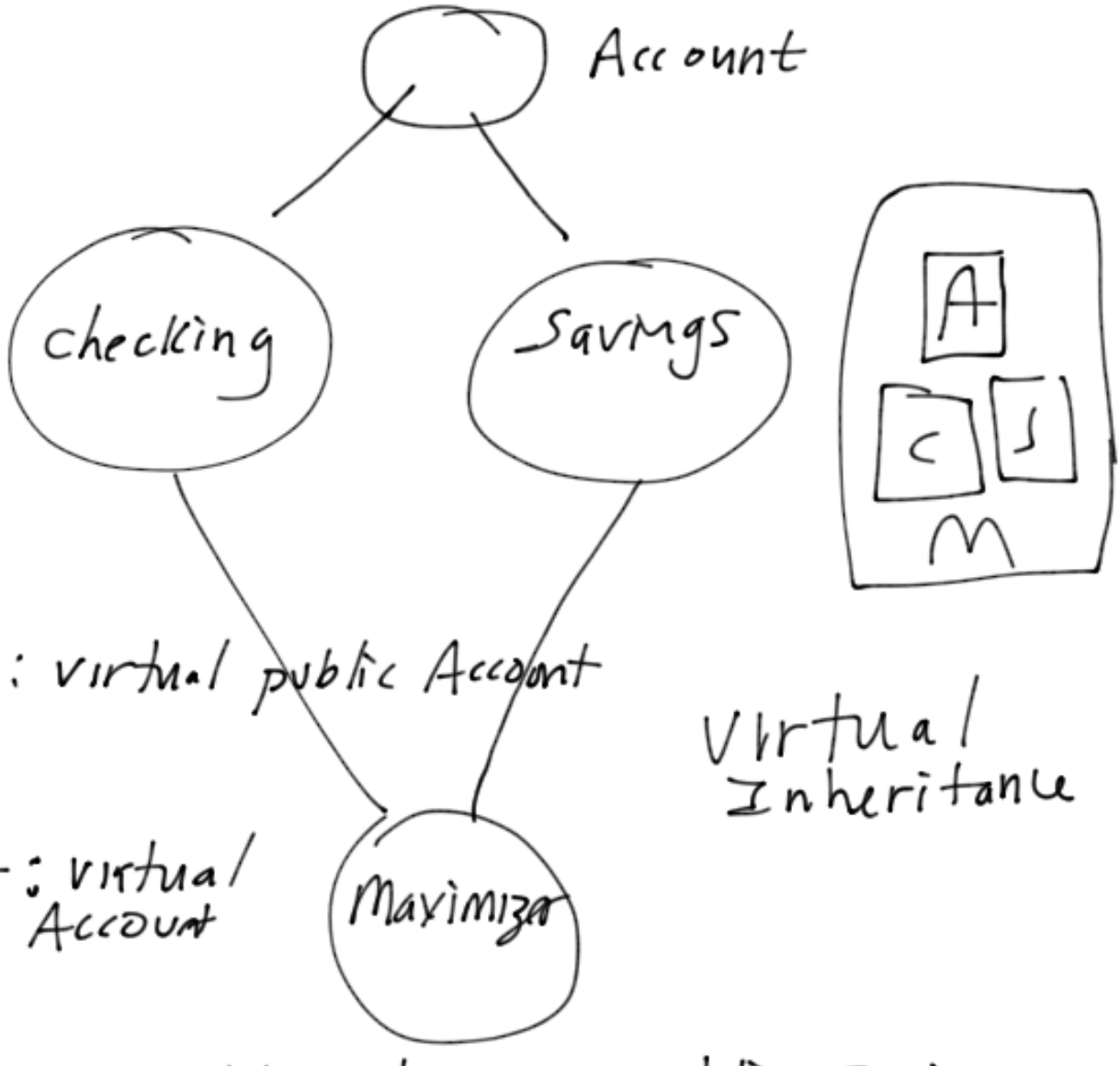


Result. Client can't call display. Ambiguous!

Solution




```
int maximizer::
    display(int value)
{
    savings:: display(value);
        :
}
```




```
class checking : virtual public Account
{ ⋮
};
class Savings : virtual
      public Account
{ ⋮
};
class Maximizer : public checking, public savings
{ ⋮
};
```

Virtual Inheritance

```
checking::checking(float rate) : Account(rate)
{
    Account obj(rate);
}
```

initialization list

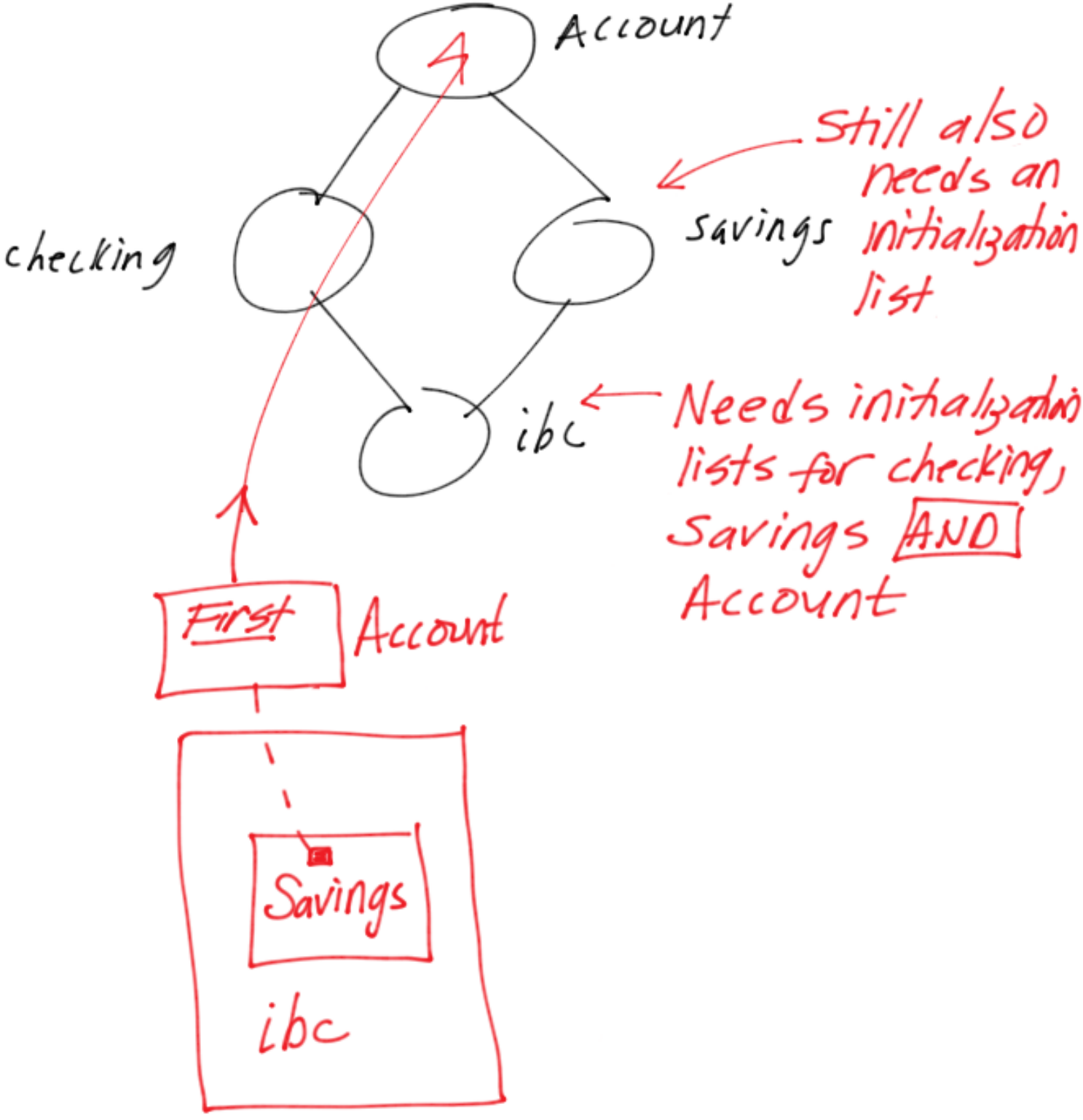
JUST a Local variable

```
#include <iostream>
using namespace std;
```

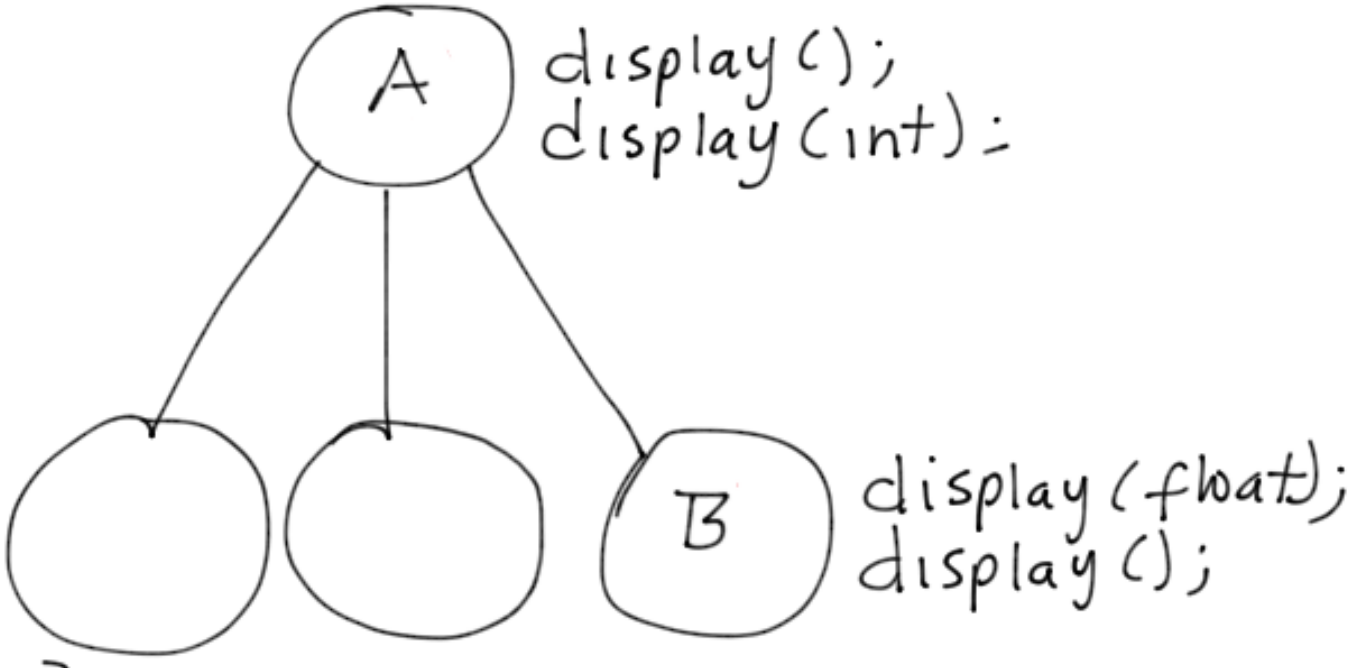
std::cin >> black;

```
class account
{
};
```

```
class savings: virtual public
    account
{
};
class checking: virtual public
    account
{
};
class ibc : public savings,
            public checking
{};
```


Account
checking
savings
ibc
Still also needs an initialization list
Needs initialization lists for checking, Savings AND Account
First
Account
Savings
ibc

Function Overloading - Revisited

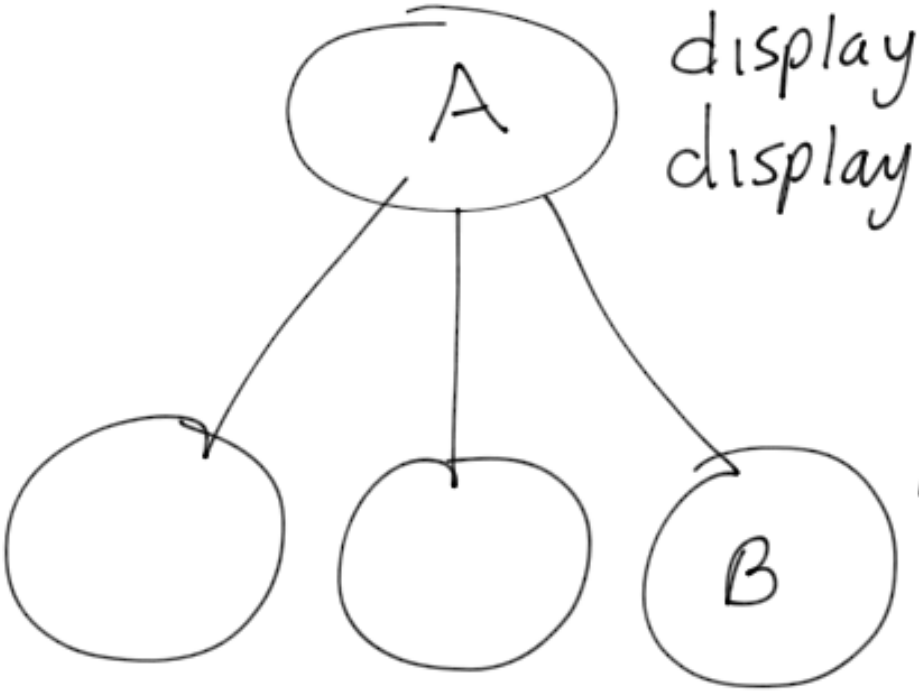


which one?

1. B obj; obj.display(); ?
2. A obj; obj.display(); ?
3. B obj; obj.display(integer); ?

   obj.A::display(integer); ?

Slight change :



display();
display(int);

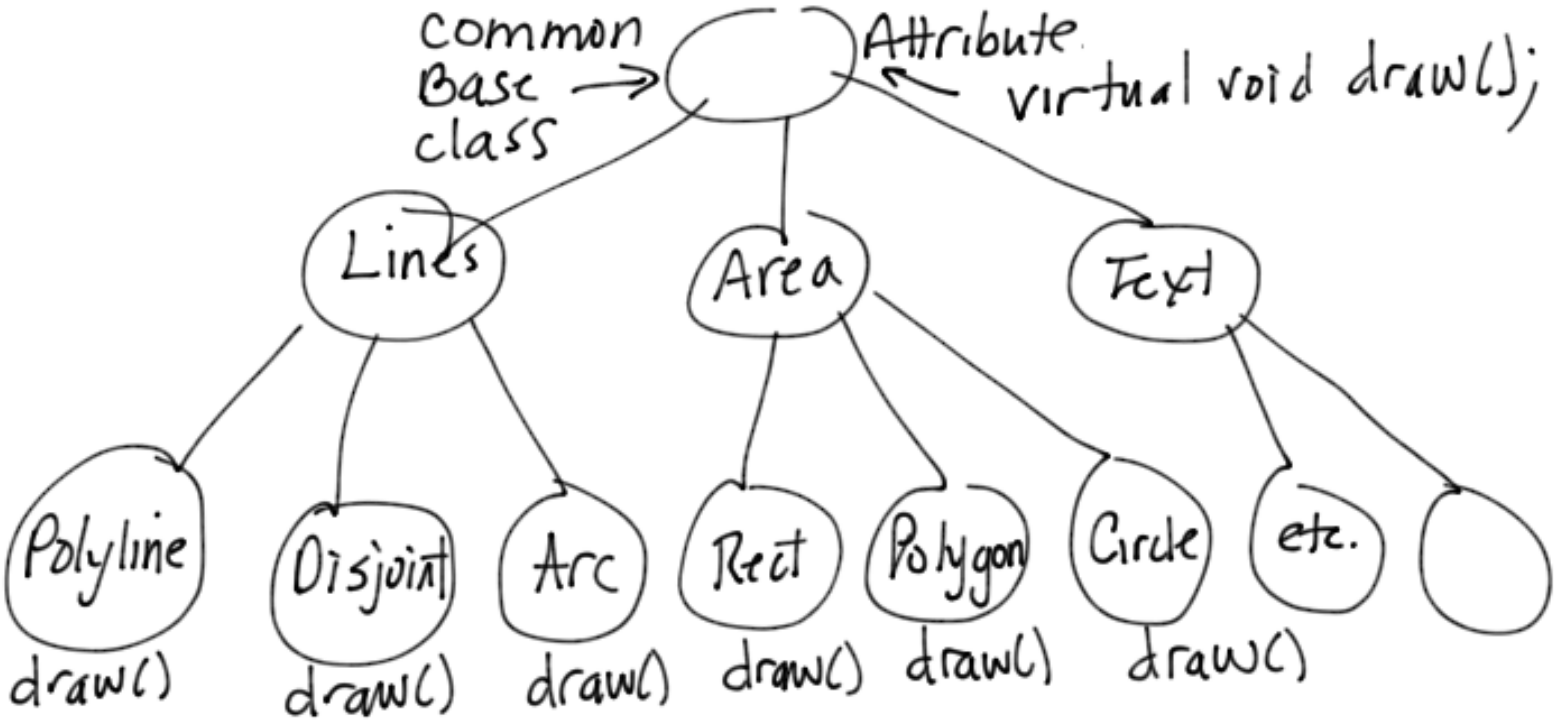
using A :: display;
display();
display(float);

Brings all unique versions of A's display within scope.

Now

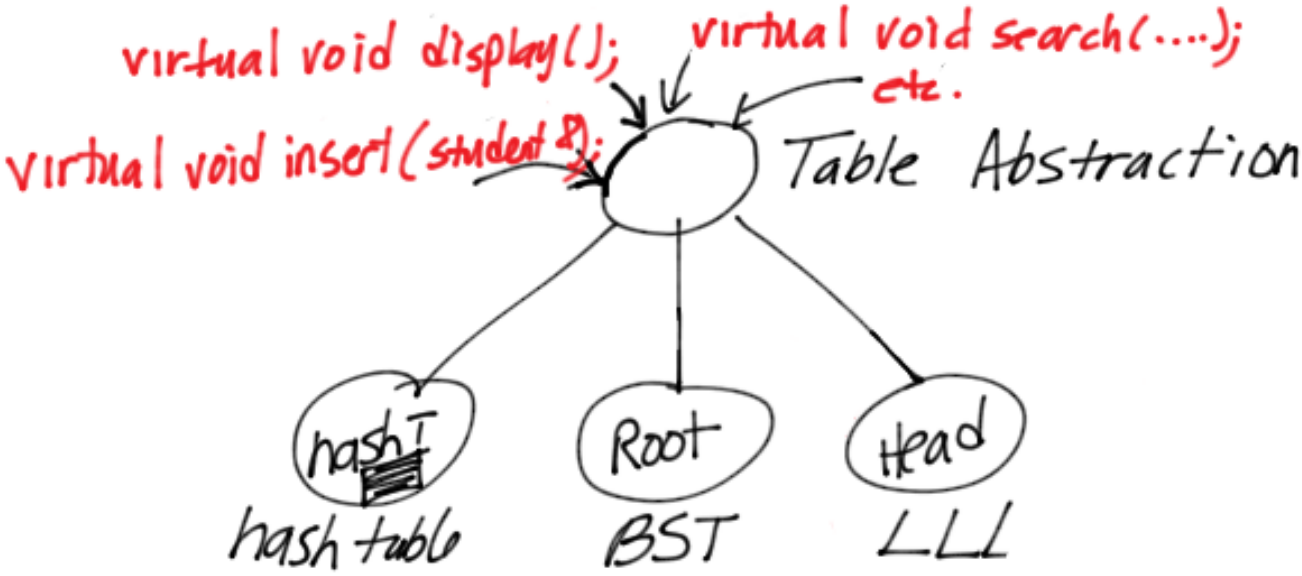
1. B obj; obj.display(); ?
   obj.display(integer);
2. A obj; obj.display();

Dynamic Binding Intro (Preview)



WITH <u>ONE</u> line of code the desired function will be called based on where we are <u>pointing</u> or referring to. Attribute *ptr = new circle;

ptr → draw();




```
Table * ptr = new BST;

ptr   insert(student);

ptr = new LLL;
ptr->insert(student);
```

Dynamic Binding!