Today CS202 - Lecture #3

1. Lecture slides - Topic #4
   - Static Binding
   - Dynamic Binding
   - Proper use of keyword "virtual"
2. Abstract Base classes & pure virtual functions
3. Programming with dynamic binding

Announcements:

1. Program #1 is due 10/14/2011 for full credit
2. Program #2 covers dynamic binding
3. Make sure to participate & submit Activities!

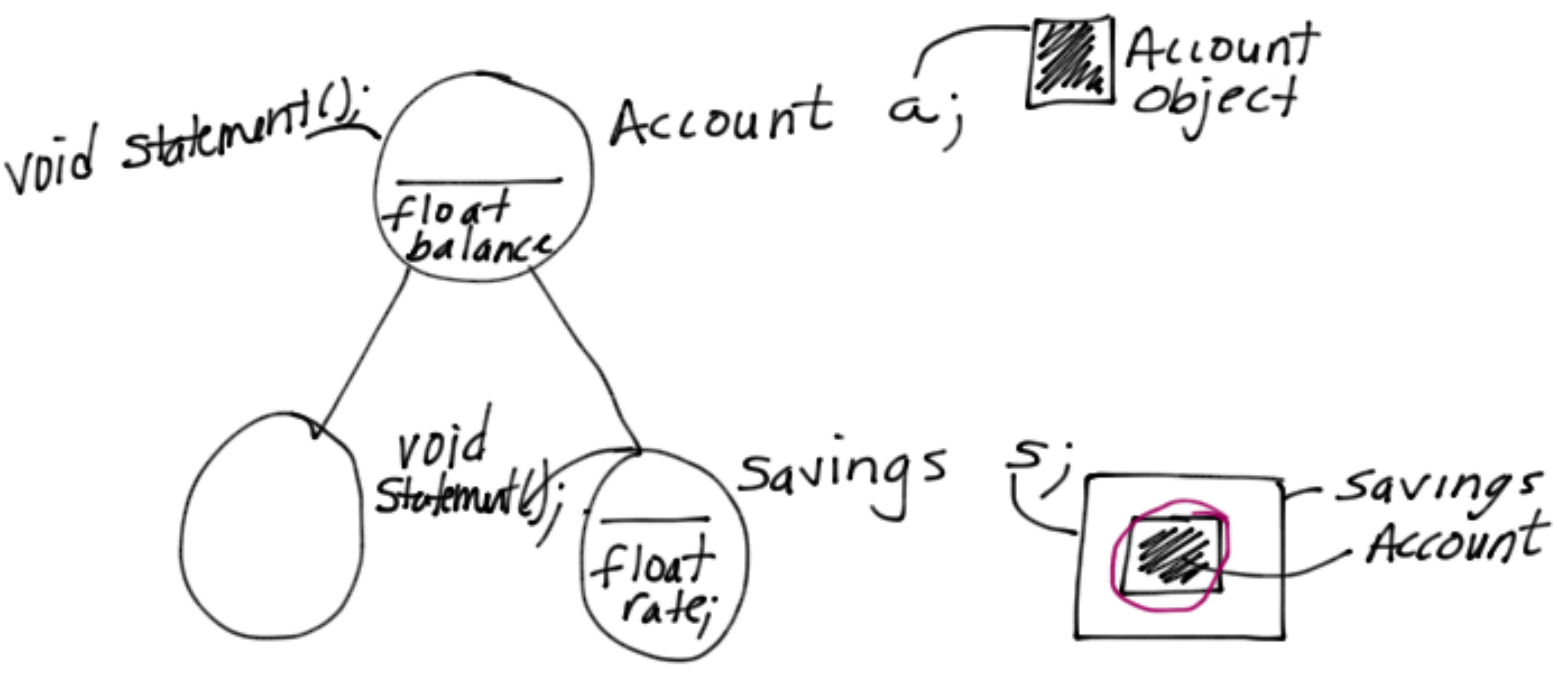
Important

- Remember, with a hierarchy **all** derived class objects are not only an object of their class but they **ARE** also an object of their base class

- This means that a derived class object can be used in **ANY** context that expects a base class. why? because a derived object **IS** at its fundamental core a base class object.

** This can't be done with "has a" relationships & represents the real beauty of inheritance !

Static Binding



① object.statement();

↑ uses the data type of the object to determine which class scope to examine for the statement function.

Examples:

a) a.statement(); ? account

b) s.statement(); ? Savings

c) a = s; ? copy just Account piece

a.statement(); ? account

Static Binding

- Uses the data type of the object OR pointer to determine which scope to find the function.
- If the function name is not defined in that scope then it will examine the parent's scope. ("has a" relationships do not do this, which is another advantage of hierarchies!)

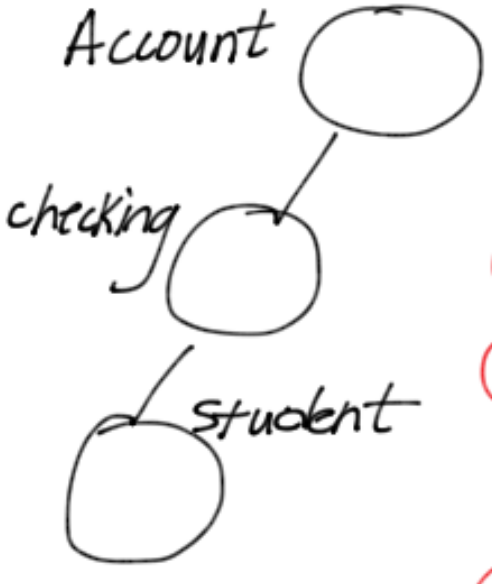
Static Binding ← what we have been doing!

ptr → Statement();

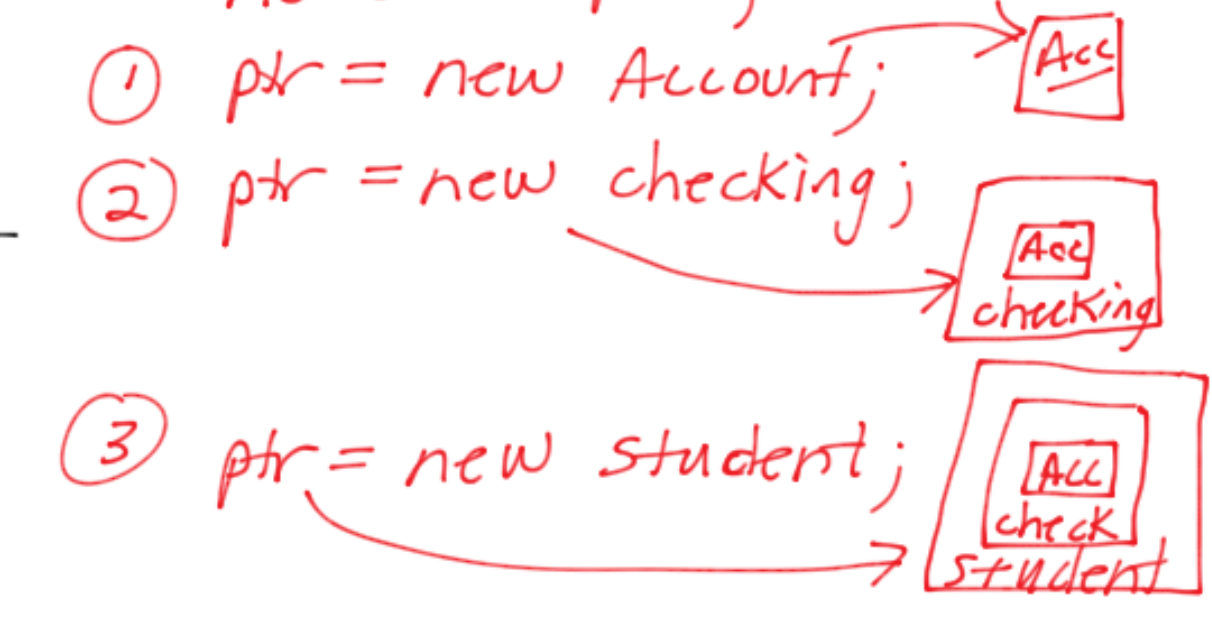
↑ uses the data type of ptr NOT what ptr is pointing to

Upcasting

1) A base class pointer can point to:
   - a base class object
   - a derived class object
     (because a derived class object IS at its inner core a base class object, plus more!




```
Account *ptr;
(1) ptr = new Account;
(2) ptr = new checking;
(3) ptr = new student;
```



This is upcasting

- No Explicit type conversion required -

OK ... so what happens when we call a member function?

ptr → statement();

Static Binding will always use the data type of your object, pointer, or reference to determine which scope to use to resolve the reference.

What about:

a) Student obj;
Account *ptr = &obj;

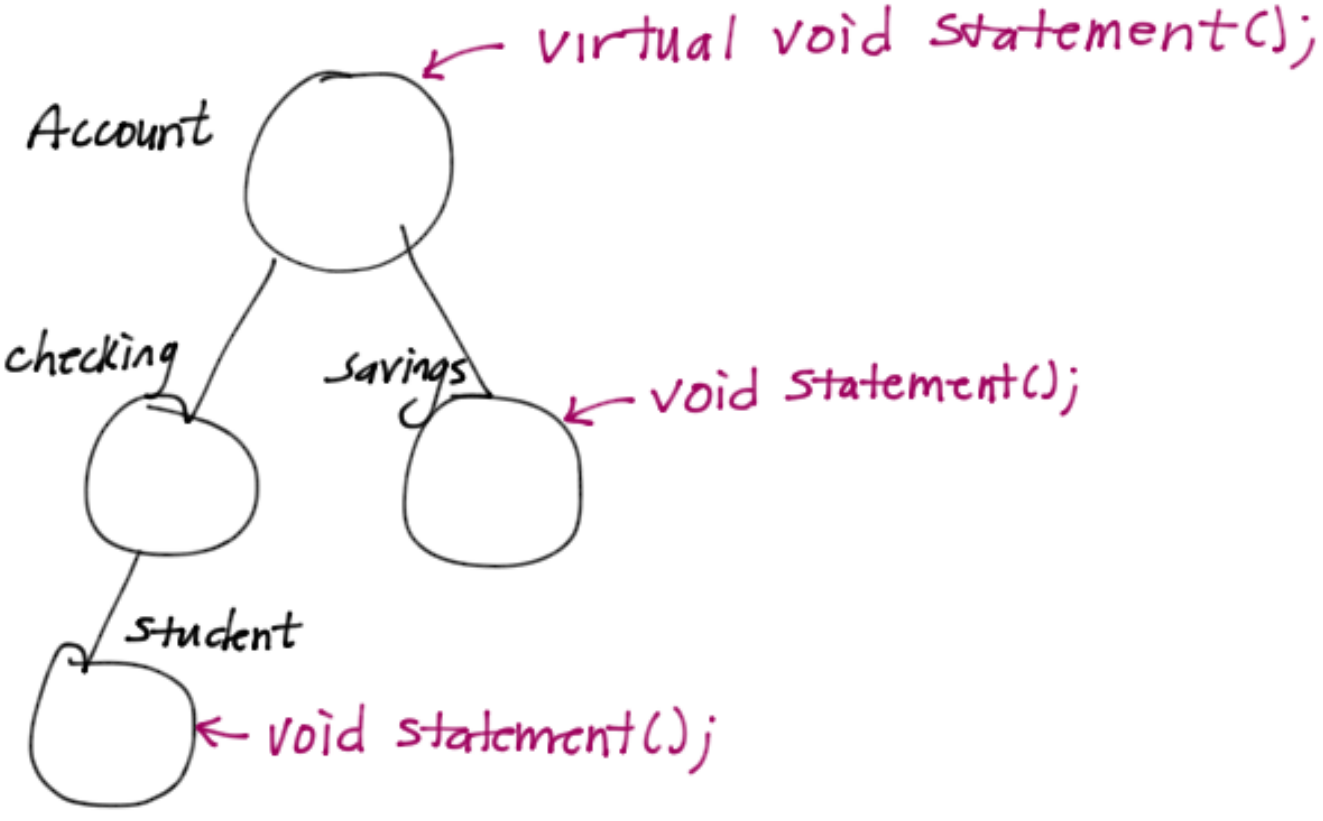
address of operator

Same problem. The Account statement function is invoked

why? static Binding

Base on the static type

Dynamic Binding



Upcasting

```
Account * ptr = new student;
ptr -> statement();
```

↑ Delays binding to the actual function until we are at run time & know where ptr is pointing.

Dynamic Binding uses the information available at run time to resolve **function CALLS**

Mixing Static & Dynamic Binding

1. The purpose of dynamic binding is to allow the application with one line of code to call one of many functions based on where its pointers or references are pointing to **AT RUN TIME**

2. Therefore the application may in fact not be aware of where the pointer or reference is pointing to at any given moment. (The function call might happen inside of a function and is unaware of the data type to which the pointer is physically pointing to.

3. Remember a base class pointer (or reference) can point anywhere within a hierarchy due to upcasting

For Example –

```
void display(Account & ref)
{
    ref.statement();
}
```

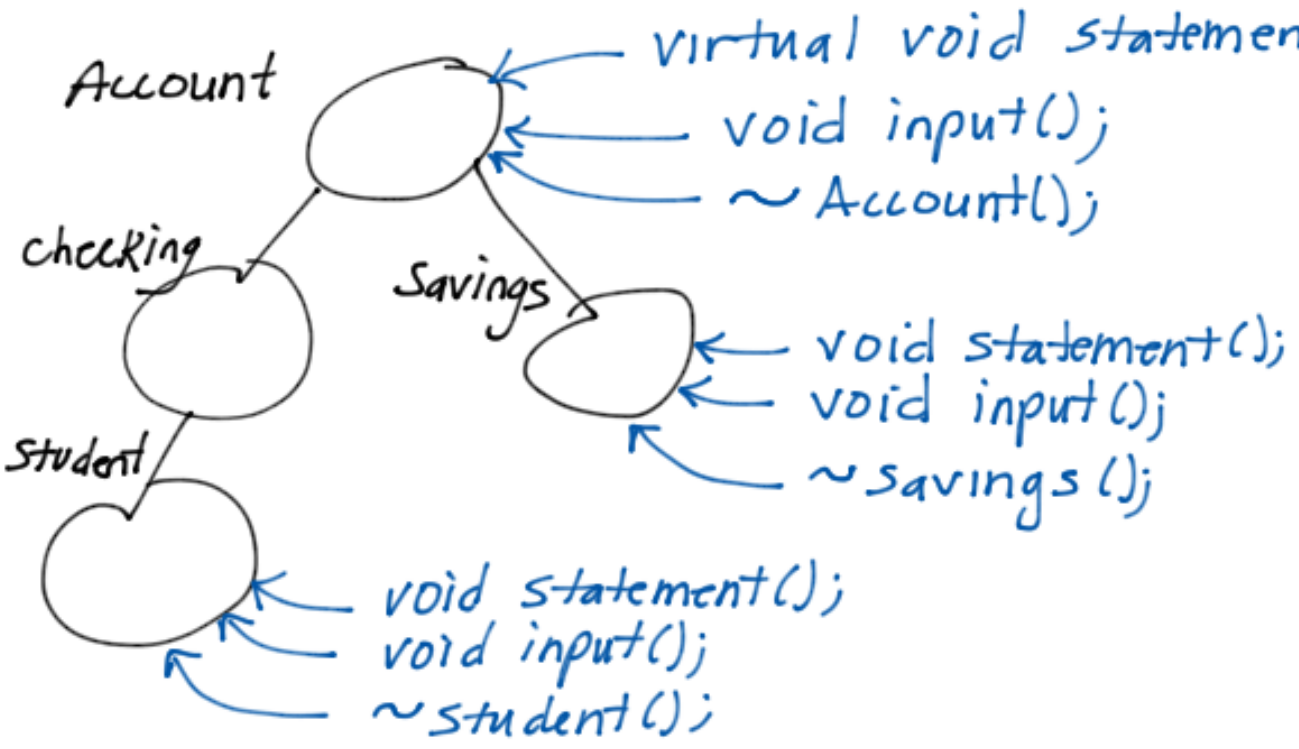
← which statement function is invoked?

a.
```
Savings obj;
display(obj);   // Savings statement
```

b.
```
Account *ptr = new checking;
display(*ptr);   // checking statement
```

c.
```
Account base;
display(base);   // Account's statement
```

So now what happens?

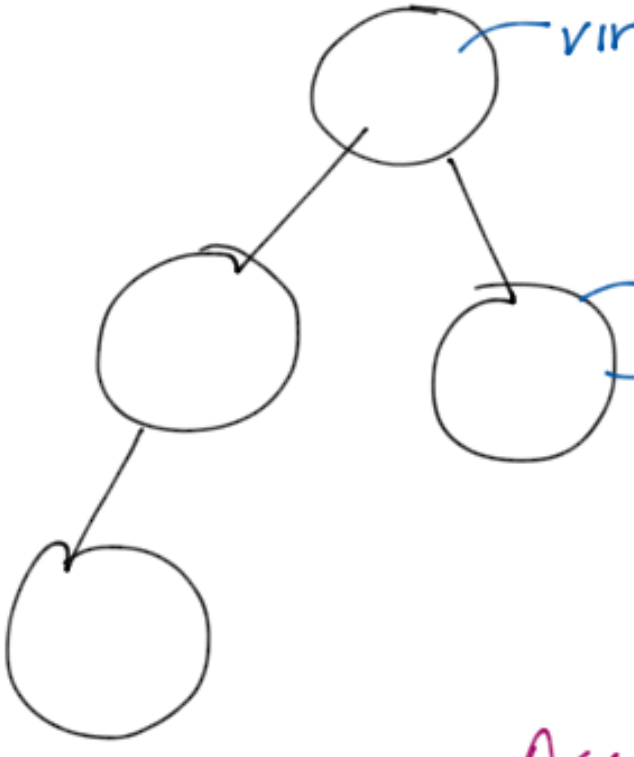


```
Account *ptr = new student;
ptr -> statement();   ?
ptr -> input();       ?
delete ptr;           ?  //wrong one! (Account only)
```

Be careful, the destructor in the base class should always be virtual when dynamic binding is used — otherwise the correct destructor will not be invoked.

What about Function Overloading vs Dynamic Binding

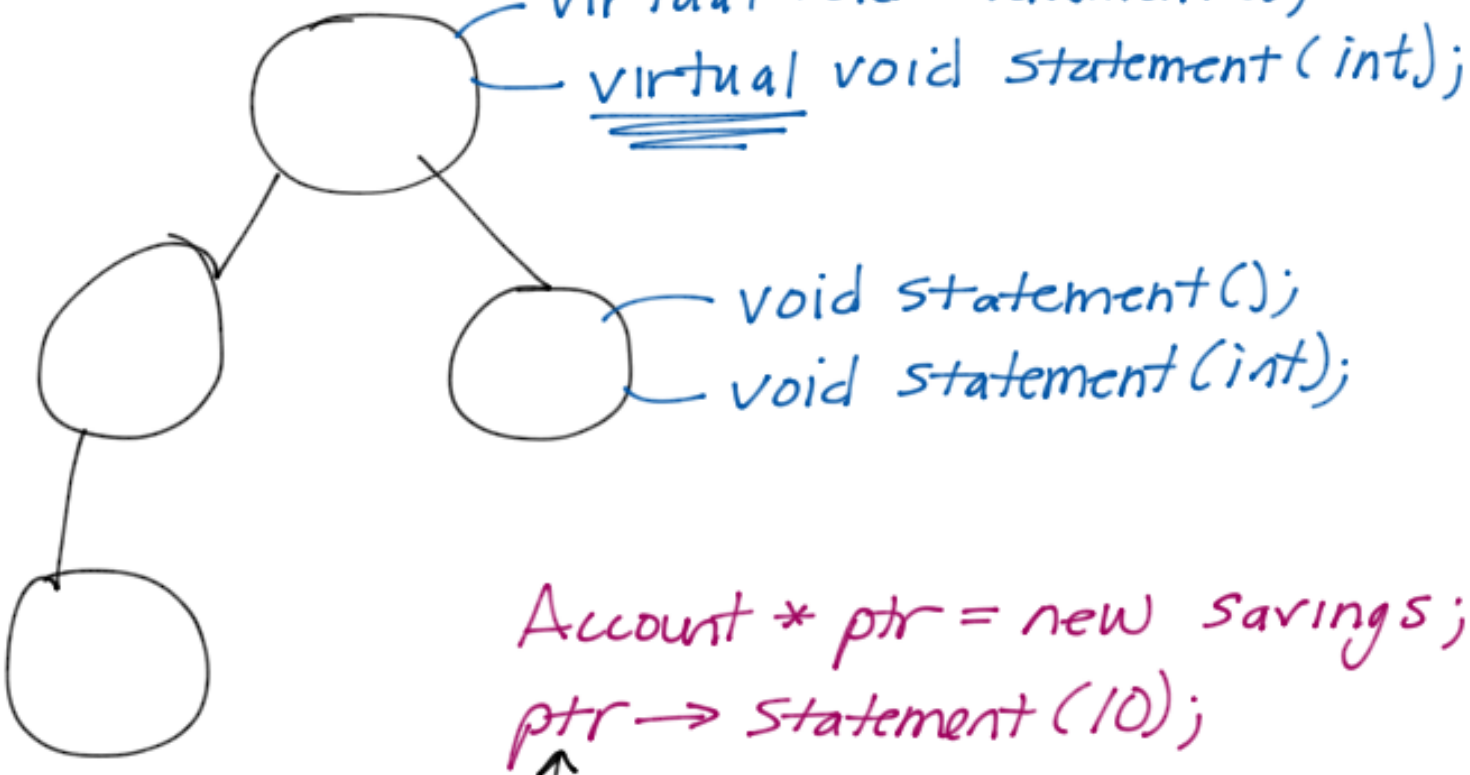
- Function Overloading is a Compile time concept
- Dynamic Binding waits to bind an object to a function call until we are running the program




```
Account *ptr = new Savings;
ptr -> Statement(10);
```

Syntax Error.
At compile time we use the DATA TYPE of the object to determine the scope

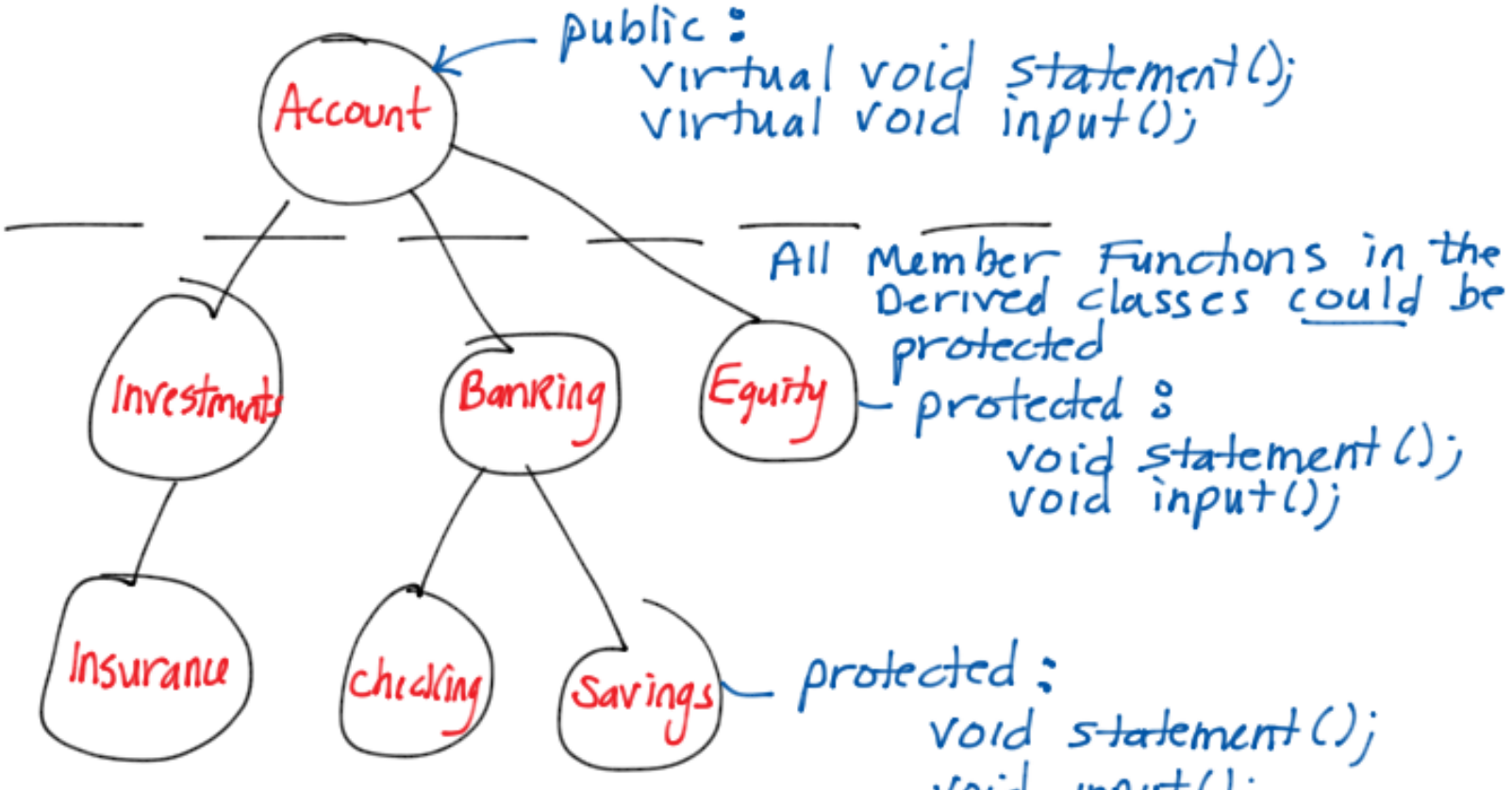
Solution ...




```
Account * ptr = new Savings;
ptr -> statement(10);
```

The compiler will use the data type of ptr to check to see if the function exists. Since it is declared to be virtual, the binding is delayed until run time and the Savings function is called (with the int arg).

Using Dynamic Binding




```
Account *ptr = new savings;
ptr -> Account :: statement();
```

} OK. Calls the Saving's statement function

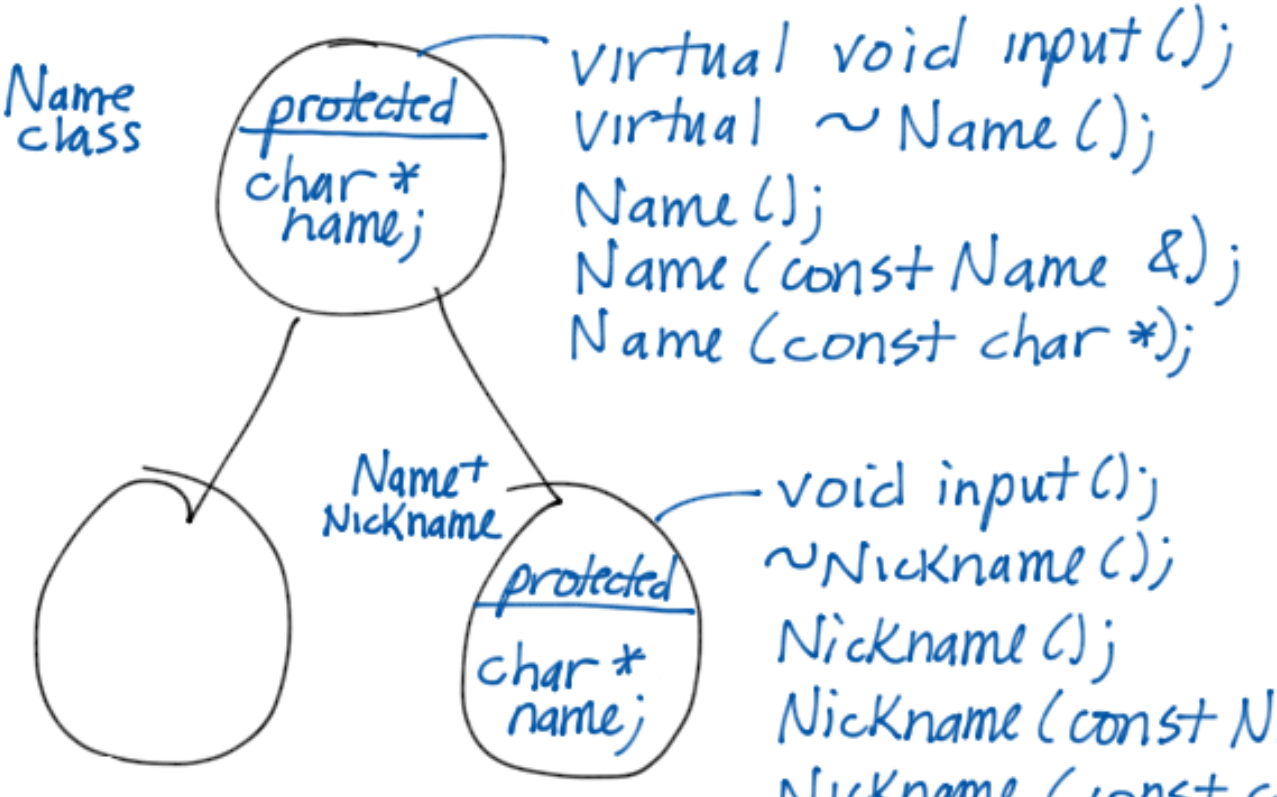
```
Savings obj;
obj. statement();
```

} syntax error!

```
Account *ptr = &savings_object;
ptr -> statement();
```

?

Member Functions vs. Data Members



What if....

① Data members were public:

```
Name *ptr = new Nickname("Sue Smith");
cout << ptr->name <<endl;
```

what would be displayed?

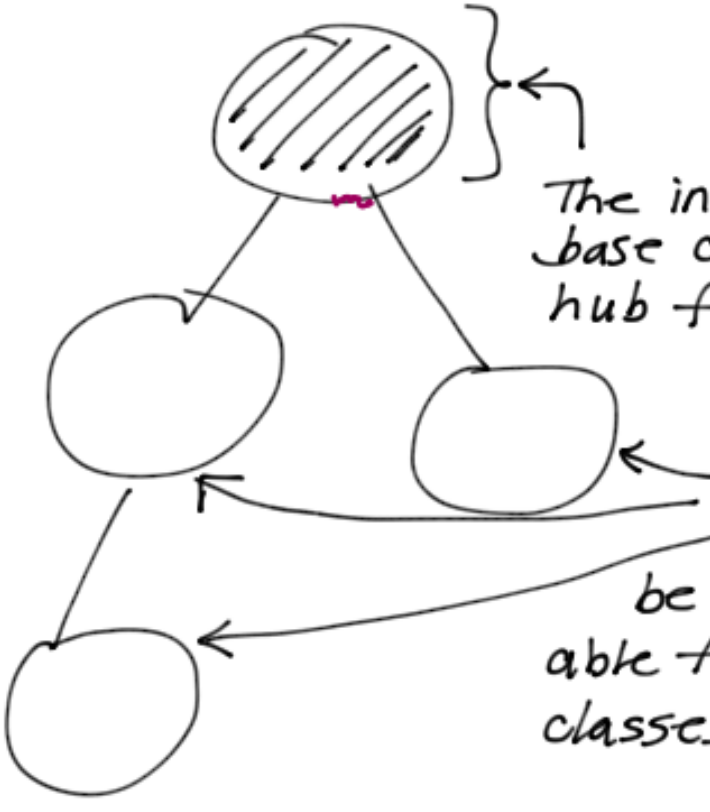
** Dynamic Binding has NOTHING to do with the data. It is about the binding of FUNCTIONS to objects/pointers/references.

Abstract Base classes

- use "Pure Virtual Functions" in the most indirect base class

virtual void statement() = 0;

this makes the prototype pure virtual

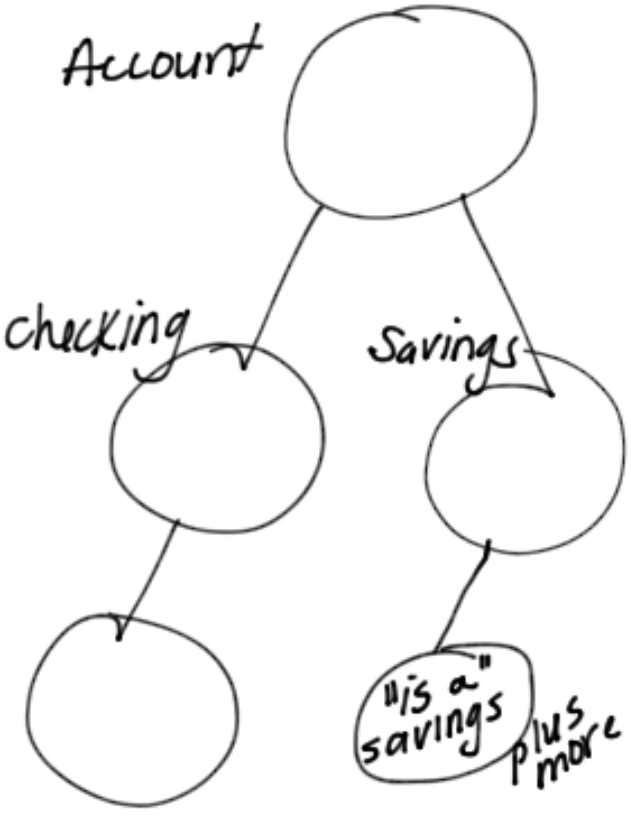


The indirect base class now becomes the hub for the hierarchy

all functions specified as pure virtual in a base class MUST be implemented if we want to be able to create objects of these classes.

1. Any class with a pure virtual function is called an ABSTRACT BASE CLASS
2. We CAN'T create objects of abstract base classes
3. They CAN have data members!
4. Derived classes MUST implement the functions OR be abstract!

Down Casting – Part of RTTI




```
Account *ptr;
ptr->function();
```

Fine if the function is virtual in the base class

what if we need to call a statically bound function? We can only do this by casting the ptr to the correct data type to which the function exists

```
Savings *result = dynamic_cast<Savings *>(ptr);
```

returns zero if ptr is NOT pointing to a savings object (or an object that IS a savings object.