Multiple Inheritance in OOP

Python supports the concept of a subclass inheriting attributes from multiple base classes, a language feature called *multiple inheritance*.

Suppose that we have a `SavingsAccount` that inherits from `Account`, but charges customers a small fee every time they make a deposit.

```
>>> class SavingsAccount(Account):
        deposit_charge = 2
        def deposit(self, amount):
            return Account.deposit(self, amount - 
self.deposit_charge)
```

Then, a clever executive conceives of an `AsSeenOnTVAccount` account with the best features of both `CheckingAccount` and`SavingsAccount`: withdrawal fees, deposit fees, and a low interest rate. It's both a checking and a savings account in one! "If we build it," the executive reasons, "someone will sign up and pay all those fees. We'll even give them a dollar."

```
>>> class AsSeenOnTVAccount(CheckingAccount, 
SavingsAccount):
        def __init__(self, account_holder):
            self.holder = account_holder
            self.balance = 1           # A free dollar!
```

In fact, this implementation is complete. Both withdrawal and deposits will generate fees, using the function definitions in`CheckingAccount` and `SavingsAccount` respectively.

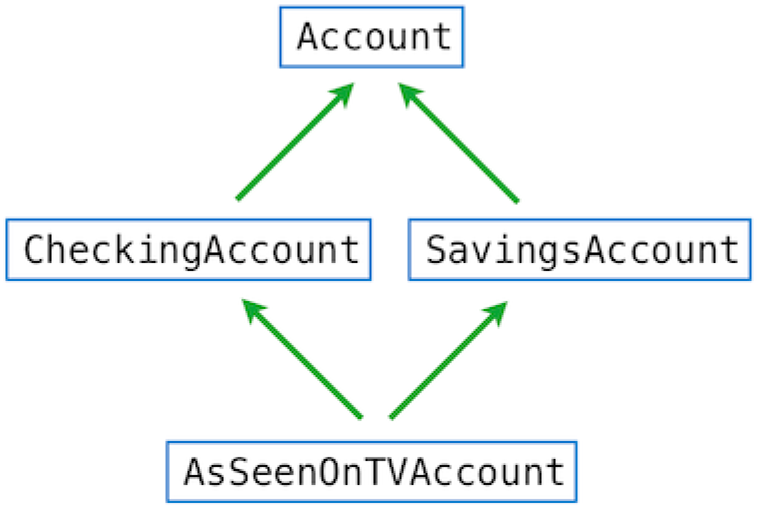
```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)            # $2 fee from 
SavingsAccount.deposit
19
>>> such_a_deal.withdraw(5)            # $1 fee from 
CheckingAccount.withdraw
```

```
13
```

Non-ambiguous references are resolved correctly as expected:

```
>>> such_a_deal.deposit_charge
2
>>> such_a_deal.withdraw_charge
1
```

But what about when the reference is ambiguous, such as the reference to the `withdraw` method that is defined in both `Account`and `CheckingAccount`? The figure below depicts an *inheritance graph* for the `AsSeenOnTVAccount` class. Each arrow points from a subclass to a base class.



For a simple "diamond" shape like this, Python resolves names from left to right, then upwards. In this example, Python checks for an attribute name in the following classes, in order, until an attribute with that name is found:

```
AsSeenOnTVAccount, CheckingAccount, SavingsAccount,
Account, object
```

There is no correct solution to the inheritance ordering problem, as there are cases in which we might prefer to give precedence to certain inherited classes over others. However, any programming language that supports multiple inheritance must select some ordering in a consistent way, so that users of the language can predict the behavior of their programs.

Further reading. Python resolves this name using a recursive algorithm called the C3 Method Resolution Ordering. The method resolution order of any class can be queried using the `mro` method on all classes.

```
>>> [c.__name__ for c in AsSeenOnTVAccount.mro()]
['AsSeenOnTVAccount', 'CheckingAccount',
'SavingsAccount', 'Account', 'object']
```

.

Source : http://inst.eecs.berkeley.edu/~cs61A/book/chapters/objects.html#multiple-inheritance