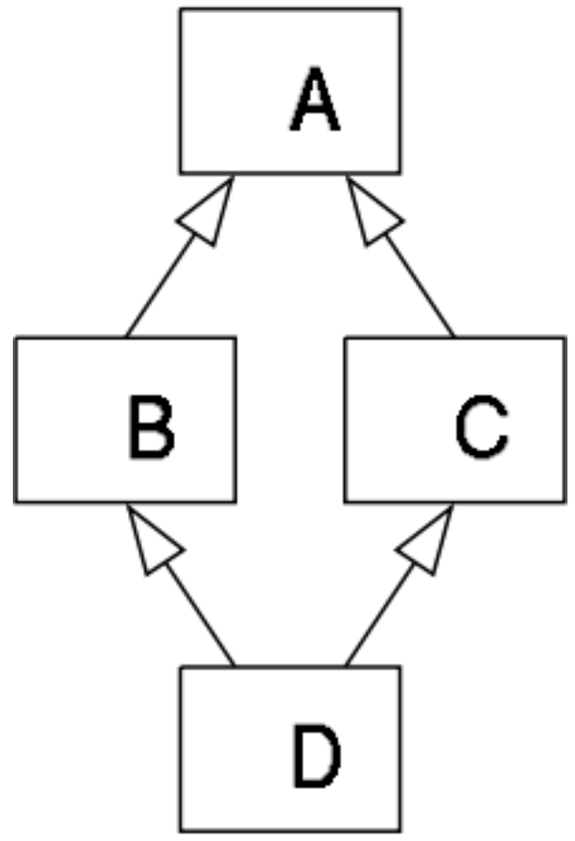
Multiple Inheritance

- **A class can inherit from more than one superclass**
  - It gets complicated

- Many languages only allow single inheritance
  - For good reason

- The Deadly Diamond of Death (DDD)
  - AKA - De Deadly Dimind of Det (DDDD)
  - A implements method m1()
  - B overrides method m1()
  - What version of m1() does D inherit



Deadly Diamond of Death

```
class BaseClass():
    def m1(self):
        print("This is m1 from BaseClass")

class SubLeft(BaseClass):
    def m1(self):
        print("This is m1 from SubLeft")

class SubRight(BaseClass):
    def m1(self):
        print("This is m1 from SubRight")


class BottomClass(SubLeft,SubRight):
    pass

class OtherBottomClass(SubRight,SubLeft):
    pass
```

```
x = BottomClass()
x.m1()
This is m1 from SubLeft

y = OtherBottomClass()
y.m1()

This is m1 from SubRight

OtherBottomClass.mro()
Out[15]:
[__main__.OtherBottomClass,
 __main__.SubRight,
 __main__.SubLeft,
 __main__.BaseClass,
 object]
In [16]:
BottomClass.mro()
Out[16]:
[__main__.BottomClass,
 __main__.SubLeft,
 __main__.SubRight,
 __main__.BaseClass,
 object]
```

- Order super classes listed determines behaviour
- **`mro()`** method tells us the method resolution order

Mugs Game:
Cannot be depending on order to determine behaviour

Multiple Inheritance example

```
class Address():
    def __init__(self, street, city):
        self.street = str(street)
        self.city = str(city)
    def show(self):
        print(self.street)
        print(self.city)

class Person():
    def __init__(self, name, email):
        self.name = str(name)
        self.email= str(email)
    def show(self):
        print(self.name + ' ' + self.email)

class Contact(Person, Address):
    def __init__(self, name, email, street, city):
        Person.__init__(self, name, email)
        Address.__init__(self, street, city)
    def show(self):
        Person.show(self)
        Address.show(self)
        print()
```

Exercise:
Draw the UML diagram for these three classes

Using the Contact class

```
class Notebook():
    def __init__(self):
        self.people = dict()
    def add(self, name, email, street, city):
        self.people[name] = Contact(name, email, street, city)
    def show(self, name):
        if name in self.people:
            self.people[name].show()
        else:
            print('Unknown', name)
```

```
notes = Notebook()
notes.add('Alice', 'alice@gmail.com', 'Cross St', 'Dublin')
notes.add('Brian', 'brian.c@tcd.ie', 'New St', 'Cork')

notes.show('Alice')
notes.show('Carol')
```

```
Alice alice@gmail.com
Cross St
Dublin

Unknown Carol
```

Exercise:
Add Notebook to the UML diagram

Posessions Example

```
help(Dog)

Help on class Dog
in module __main__:

class Dog(Pet)
 |  Method resolution order:
 |      Dog
 |      Pet
 |      Mammal
 |      Animal
 |      Posession
 |      builtins.object
 |
```

```
class Animal():
    def __init__(self,name):
        self.name = name

class Mammal(Animal):
    pass

class Posession():
    def __init__(self,value=0):
        self.value = value
    def get_value(self):
        return self.value

class Pet(Mammal,Posession):
    def __init__(self, name, value):
        Mammal.__init__(self, name)
        Posession.__init__(self, value)

class Dog(Pet):
    def __init__(self,name,value,chipNo):
        self.chipNo = chipNo
        Pet.__init__(self, name, value)

class Cat(Pet):
    pass
```

Possessions Example

```
katyPurry = Cat("Katy Purry",5)
katyPurry.value, katyPurry.name
Out[38]:
(5, 'Katy Purry')
In [39]:
winnie = Dog('Winnie the Poodle',500,991199)
In [40]:
winnie.__dict__
Out[40]:
{'chipNo': 991199, 'name': 'Winnie the Poodle', 'value': 500}
```

Multiple Inheritance - Final Comments

- Probably safest to steer clear
- Method Resolution Order
  - If you need to worry about MRO you should be worried
- 'Mixins'
  - Contacts Example & Possessions Example MRO doesn't matter
  - Mixing different kinds of methods and attributes