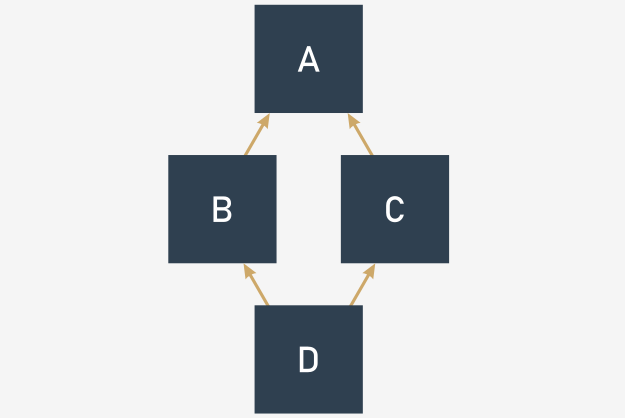
**O problema do diamante**

O segundo exemplo do espectro de questões que podem eventualmente surgir de heranças múltiplas é ilustrado por um problema clássico chamado o **problema do diamante**. O nome reflete a forma do diagrama da herança - dê uma vista de olhos na imagem:



* Existe a superclasse mais alta chamada A;
* Existem duas subclasses derivadas de A: B e C;
* e há também a subclasse mais baixa chamada D, derivada de B e C (ou C e B, pois estas duas variantes significam coisas diferentes em Python)

Consegue ver o diamante ali?

Dê uma vista de olhos ao código no editor. A mesma estrutura, mas expressa em Python.

Algumas linguagens de programação proíbem a herança múltipla, e como consequência, não o deixam construir um diamante - esta é a rota que Java e C# escolheram seguir desde as suas origens.

O Python, contudo, escolheu uma rota diferente - permite herança múltipla, e não se importa se escrever e executar um código como o que está no editor. Mas não se esqueça do MRO - é sempre ele que manda.

Vamos reconstruir o nosso exemplo da página anterior para o tornar mais parecido com um diamante, tal como abaixo:

class Top:

def m\_top(self):

print("top")

class Middle\_Left(Top):

def m\_middle(self):

print("middle\_left")

class Middle\_Right(Top):

def m\_middle(self):

print("middle\_right")

class Bottom(Middle\_Left, Middle\_Right):

def m\_bottom(self):

print("bottom")

object = Bottom()

object.m\_bottom()

object.m\_middle()

object.m\_top()

Nota: ambas as classes Middle definem **um método com o mesmo nome**: m\_middle().

Introduz uma pequena incerteza na nossa amostra, embora estejamos absolutamente certos de que pode responder à seguinte pergunta-chave: qual dos dois métodos m\_middle() será realmente invocado quando a seguinte linha for executada?

Object.m\_middle()

Por outras palavras, o que verá no ecrã: middle\_left ou middle\_right?

Não precisa de se apressar - pense duas vezes e tenha em mente o MRO do Python!

Está preparado?

Sim, tem razão. A invocação ativará o método m\_middle() , que vem da classe Middle\_Left . A explicação é simples: a classe está listada antes Middle\_Right na lista de herança da classe Bottom . Se quiser ter a certeza de que não há dúvidas, tente trocar estas duas classes na lista e verifique os resultados.

Se quiser experimentar algumas impressões mais profundas sobre a herança múltipla e pedras preciosas, tente modificar o nosso snippet e equipar a classe Upper com outro espécime do método m\_middle() , e investigue cuidadosamente o seu comportamento.

Como pode ver, os diamantes podem trazer alguns problemas à sua vida - tanto os verdadeiros como os oferecidos pelo Python.