Ereditarietà multipla

Linguaggi come C++ consentono ad una classe di estendere più classi per ridurre la duplicazione di codice:

```
class Veicolo { ... }

class VeicoloTerrestre : public Veicolo { ... };
class VeicoloAereo : public Veicolo { ... };

class Rifornibile { ... };

class Automobile : public VeicoloTerrestre, public Rifornibile { ..
class Aeroplano : public VeicoloAereo, public Rifornibile { ... };
```

le interfacce non risolvono il problema, visto che richiedono ad ogni implementatore di fornire il codice dei metodi.

Problematiche

- possibilità di ereditare metodi con segnatura identica ma implementazione diversa da due classi distinte (se la segnatura è diversa si risolve con overriding);
- *diamond problem*: ereditare da due classi che hanno una superclasse in comune può portare a variabili d'istanza e metodi duplicati.

Soluzioni:

C++ *vtable* multiple per ogni classe, sintassi per la disambiguazione, distinzione tra metodi/classi `virtual` e non; maggiore controllo ed efficiente di default, ma richiede consapevolezza da parte del programmatore;

Java gestione delle interfacce trasparente al programmatore (ricerca a runtime nella *itable*), ma uso limitato dei meccanismi di ereditarietà multipla (introdotta parzialmene tramite implementazioni di default nelle interfacce, ma il compilatore vieta gerarchie ambigue).

Python linearizzazione della gerarchia delle classi (che con ereditarietà multipla è un grafo invece di un semplice albero) con algoritmo C3

Dart/Scala composizione di funzionalità (mixin) ed ereditarietà singola.

Le soluzioni di Java e C++ richiedono un compilatore in grado di individuare staticamente le scelte possibili per metodi ambigui, e per generare la struttura dei dati a runtime (vtable/itable) serve conoscere tutti i metodi di una classe (non si possono aggiungere dinamicamente a runtime); per questo Python è costretto ad adottare un approccio meno efficiente.