The Problem with Single Inheritance.

After having reviewed the text, and thinking about our conversation, I'd like to outline one of the major problems with only having single inheritance. Please looked at the attached spreadsheet. This is an example of a class structure you may see with Vehicles. It seems clear to separate land and water vehicles, and then further define classes for Cars vs. Trucks, etc. We can imagine some methods that all Land Vehicles would share, such as “drive”, or “honk” or “shift gears”. The Same with Boats, “drop anchor”, “radio shore”, etc. All classes would probably share things like “turn vehicle”, “turn on”, “shut off”, etc.

However, what if we wanted to include a method such as “Accept Non-driving passenger.” We can see in the land Vehicle class alone that such a method would be vital for the Taxi and Emergency Vehicle classes, but probably not needed for Moving Truck class and not at all wanted by the Race car class. We could move classes around and instead define “Passenger moving cars” and “non-passenger cars”, but our “Accept Non-driving passenger” method would also be necessary for the Ferry class and probably the Fishing Boat class. So where would we put this method? The only option seems to be the “Vehicles” Superclass, but then that method would be inherited by classes that don't need it (Race car class, Freighters class.)

How about a method to give a manifesto of all contents aboard a vehicle? That is something the Freighter and Moving Trucks classes would need, but Taxi, Race Car, Fishing Boat classes wouldn't.

Or how about an instance variable that tracks the current weight of ONLY the contents aboard the vehicle? Again, Ferry and Moving Truck classes, and maybe Race Car classes well, but probably not Taxi or Fishing Boat classes.

Those methods would have to be defined in a Superclass. But then we are left with a bloated superclass Vehicle that contains a lot of methods that classes that are subclassing from it don't need. Overall, the problem with single inheritance shows up when some classes that are seemingly unrelated on a class tree want to share some type of behavior. With only a single class to inherit from, we have to put these related behaviors somewhere up the tree where it will likely be inherited by classes that do not need it. We can try to alter our inheritance structure, but we will always be left with some classes that only want to share some behaviors, but are otherwise unrelated.

Ruby of course tries to solve this problem by allowing as many Modules to be mixed in to a class as is necessary.