Skill Matrix

My goal is to create a basic prototype of the skill matrix in C#. This is an exercise in my C# for D2L interview, so I want to make sure I find reasons to use abstract classes, recursion. I’ll reproduce the concept of a database without using a database.

Some basic deliverables I have in mind:

* **Stores actual data (least amount work necessary to accomplish).**
* **Uses one abstract class for all my “table classes” to inherit from**
* **make the crappiest skill dependency I can, as quick as I can**
* That’s it, stop there. No more features allowed. The minimum tables allowed for this.
* Feature two: use recursion to retrieve skills (come up with some kind of skills dependency)
* Undecided: A validation process … hinders my practice abstract class goal… But I’ll imagine it will be in place for now and later.
* accept that this console-based approach is less usable.

out of scope but cool features

* an arbitrary skill size limited to 10ish, at which point they must appear under a sub category… sort of like a rebalancing of a hash table that executes automatically when a particular grouping exceeds a specified number. however the automatic selection of a parent name is unfortunately beyond my skill atm. out of scope.
* use auto complete as users type in a field. this would make this way more usable … well that’s a shiiiit realization for my C# goal... I suppose that’s way outta line now. so that’s an argument for doing this non-console, and probably it’ll be way easier to find code that does this for web than not. out of scope.
* mutually exclusive skills. fuck that noise, waaaay out of scope, waaaaay too difficult.
* shit, I wanna do a tagging system. basically the goal with this consideration is to have a shortcut to circumvent clicking / digging deep into a hierarchy to find a specific skill. ugh. it’s very pleasing an idea to consider, but I must put this out of scope. No Tagging.

So next what I’d like to focus on is resolving some questions I have. So right now I have the following hierarchy: Table > Skill > SkillHave, as well as Table > Person > Member. Now the functionality I want is to model after SQL, so I want any of these classes to be able to invoke Add(). Currently I’m supposing I can do this with **virtual** in Table only, and **override** in all other children classes. Is this true? I clearly want Add() to perform validation logic. … specify it and try it…

* Add() should invoke the ID of that class, so I want a static variable that’s overridden?? in each child?? And supposing override works, can I apply to multiple depths?
* Add(): Also I’m looking for a way for it to “validate locally”, i.e. in order to have a SkillHave, it necessitates having a Skill. I’m leaning towards the fact the code itself handles this, there’s no additional validation I can really implement.
* OK so clearly one requirement I have is a unique static ID at each class.
* I’m thinking I may want base.Add() or something, not sure yet a reason why.
* Pondering How To: I can inherit the fact it starts at 0… but that’s not enough to be valuable
* So far I see the class itself can be abstract, as well as the methods

Since I clearly cannot have virtual and override fields, I’ll have to reconsider. I thought I could use the magic of OO to avoid writing that line in every file, but it looks like no. At least not for tonight. Hmm… But I do for sure want the functionality of a static class variable… and I still imagine there will be benefits to this inheritance relationship I’ve created, and I don’t want to rename that variable differently for each class.

so now I must consider how to avoid

OK so now having considered how to use inheritance, I’m concluding I’ll have to mix it up a little. Doing a straight inheritance line of Table > Skill > SkillHave caused a variety of issues.

Issues, Realiziations:

* calling any child in an inheritance hierarchy calls the parent constructor. originally I wanted each class level to have its own IDCount automatic increment system, which I’d placed in each level’s constructor. This was not a good idea. I’ve concluded *I can enforce inheritance of the instance field ID* by using an **abstract class Table** at the top of the chain, and that I *cannot use* **interface Table** as interfaces do not allow instance variables. An abstract class appears superior to my needs at this time, so I should fully explore that next. If I deem it a dead end, I’d use an interface next.
* My next query is to decide whether I’ll want to inherit directly at each level from the abstract Table class, or whether there’s still benefit having SkillHave inherit from Skill?... well actually yes I can do both now.
  + Skill : Table, SkillHave : Table, this way they both get the ID instance field (Are they required to implement, I’d like to check the contractedness of this…)
  + SkillHave : Skill, Table would be very interesting… is that even possible??? Try now… NOPE. As I expected
* So now that I’ve disproven multiple class inheritance at the same level, I have to decide whether to stick with Table > Skill > SkillHave or not. This does seem workable, I’ll try using this.
* OK, so now I have another dilemma. I want to have SkillHave generally include the same stuff Skill has, but I cannot do this via regular inheritance because it will call the constructor, and these are to be separately instantiatable things.
  + let’s make this more specific. I want to access the instance field ID and methods I made in the abstract Table class **(which btw I haven’t even tested if just the pure inheritance causes a separate version**), and I want to access future methods that’ll come down.
  + Now, following Panar’s OO concepts: SkillHave has a HAVE-A relationship not a IS-A relationship to Skill. Does this mean inheritance, or having an internal field of Skill within SKillHave, or something else? The proper expression of this goal is now looking to COMPOSITION (HAS-A) instead of INHERITENCE (IS-A).
  + Clearly, a SkillHave is COMPOSED of a Skill PLUS LEVELs.

consider how I want measurement to occur. Is it a class, interface, or merely an enum?

as an enum, it would be enum Measure { 0, 1, 2, 3, 4 };

as a class, it would be abstract, so I could pass on fields via inheritance. thus this would be an IS-A relationship, thus inherited. However double inheritance is prevented. I could adapt my hierarchy to be Table > Measure > Skill. Does that even work with double abstract?

I’m going to try listing below the kinds of tables I think would be useful. So far basically I’ll need a set of tables that are for data, a group that entirely would later be replaced by SQL later. Separate from this group would be operational classes that control the function of this program.

Skill

any string

level 0 - 5

desire 0 -5

Attempt at hierarchy

code

techniques

debugging

problem-solving

data structures

software (ide)

netbeans

eclipse

visual studio

2013

jetbrains

intellij idea

android studio

web storm

languages

C#

C++

Java, etc.

art

animation (2d)

character animation

animation (3d)

character animation

rigging

uv mapping & unwapping

texturing

style

pixel

cel shading

music

management

software (office)

Word

Excel

PowerPoint

Project

Visio

Google Drive Docs

team delegation