Brandon Swanson

Menu Design

TLDR: most recent implementation of Menu and Menu Item can be seen in

SwansonObjects/menu.hpp

or at https://github.com/swanyriver/SwansonObjects/blob/master/menu.hpp

As it happens I had wanted the functionality of such a menu back in week 2. I did not enjoy juggling so many different .cpp files for the exercise components and wanted a simple way to make sure they all compiled on the flip server. So I made the exercise components all in one file, with each one triggered by a menu class function. This helped to define the behavior that would be repeated by each different component. An introduction to each piece was presented and then after it was completed the menu function would give the option to repeat.

This allowed me to transfer this, intro->execute->repeat behavior easily across all my exercise component. It also made it easy to check their functionality against multiple inputs, and check them after changes to .hpp files that the included. To even more facilitate this I added a “Demonstrate All” function to the menu that would execute each item in the menu one at a time.

https://github.com/swanyriver/ASSIGNMENT2/blob/master/functionDemo.cpp#L49

This first version of the menu was not a separate class. But in week 3 I sought to encapsulate this functionality into a class in a separate .hpp that could be used in any context. It was a success because I was able to not only use it for exercise components but also able to (with no modification) include it

and use it in my project component for week 4. It has changed some this week to make its functionality even more encapsulated. I will start out by describing a basic menu design with the features we have learned this week. Then go over my design for a menu. and then discuss the evolution of my actual menu object.

Without using subclassing or abstract classes I imagine a menu would have a display and selection getting function. It would maintain a list of titles associated with number choices and then after getting input from user (and validating that it was a valid selection) would then execute the select function, offer to repeat it, and then possibly display the menu again.

Image of this simple menu design attached below

In this case a switch could be used to select which function would be called. This however is not a very portable or encapsulated object. It would be necessary to modify the code for each additional item on the menu, as well as modify the code if it were to be re-used in a different program. Ultimately a poor excuse for an object.

I wanted to come up with a way to have this menu be able to host my exercise components, (or other actions) without having to re-write the menu every time. I then found Function Pointers, I could have the Menu hold a collection of function pointers that would could be executed upon selection. And the program that was invoking an instance of this menu object could supply the functions being pointed to. And again to further encapsulate the features and functionality of these selections I created a MenuItem subclass which the Menu could hold a collection of.

Now I had a menu that had as public members its intro string, a boolean for repeating the menu, And a private list of menu items. Each menu item needed to be constructed with an intro string, a title (for the menu to display), and a pointer to a function. Each menu item had a public function called itemSelected() that would loop over displaying the intro and executing the function as long as the user desired.

So when invoking the Menu item it would be constructed with an intro, its behavior modified via its public bool members, and items added via AddItem(MenuItem) (or the “idiot constructor” add item method). Then once all the Items had been added calling ShowMenu() would display the options and request an input for selection. Upon valid input the appropriate MenuItem object’s ItemSelected() method is called.

After a little bit of use of this object there were some apparent limitations

- Functions being called had to match the same signature as the pointer member of Menu Item void()
- Specialization through subclassing was obligated to initialize the function pointer.

I made a subclass of this menu item for the automatic compiling of the exercise components. (I also used a makefile for this purpose but wanted to experiment with doing it programmatically). But in this subclass all the functionality was within the itemSelected() function and had no need to call the void() function pointer. When developing this subclass I was passing in a dummy do-nothing function. But when moving this behavior into its own .hpp it was quickly apparent that I needed a better solution.

The problem was that I was trying to specialize on top of an already specialized version of the class. So I refactored the MenuItem class and turned it into an abstract class by making the Item selected

function a pure virtual function. Then this version that calls a function pointer was one specific implementation and other versions only need to implement the function ItemSelected().

https://github.com/swanyriver/SwansonObjects/blob/master/menu.hpp#L60

Upon reflection.

It is interesting that after time I ended up moving to the interface model for the menu item. I was originally excited about the function pointer because it meant that i did not need to make every possible functionality a subclass of menu item and implement its abstract method (this would be like java Listeners).

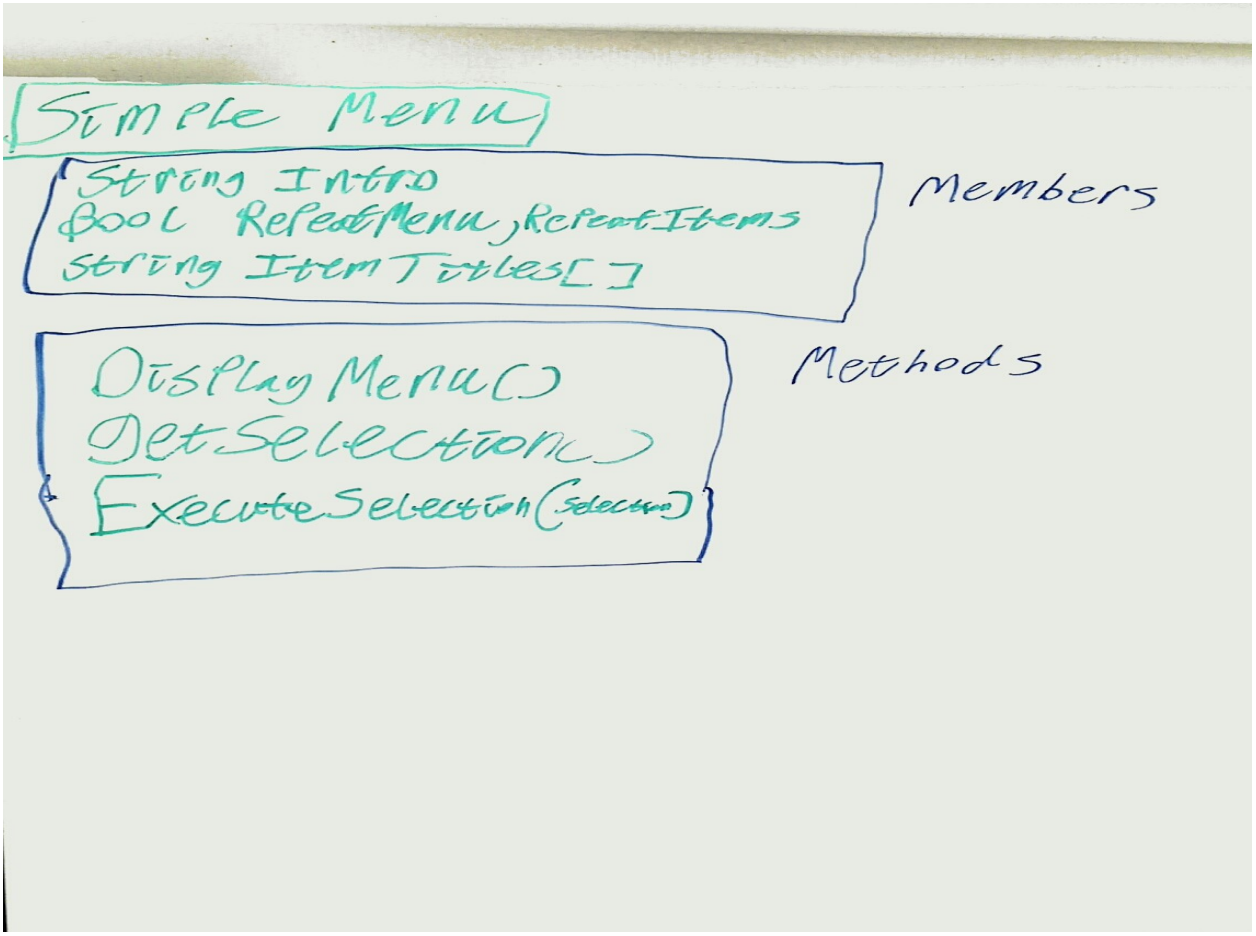
Also after switching to this abstract class i was forced to use pointers to the menu items rather than the menu actually holding references to menu item variables. This was important because pointers (apparently) are the only way to achieve object polymorphism. and also because there was no way for me to hold instances of the abstract base class. This is obvious as that class is not one that can be instantiated. This is something that was somewhat obscured when i was working with interfaces in Java.

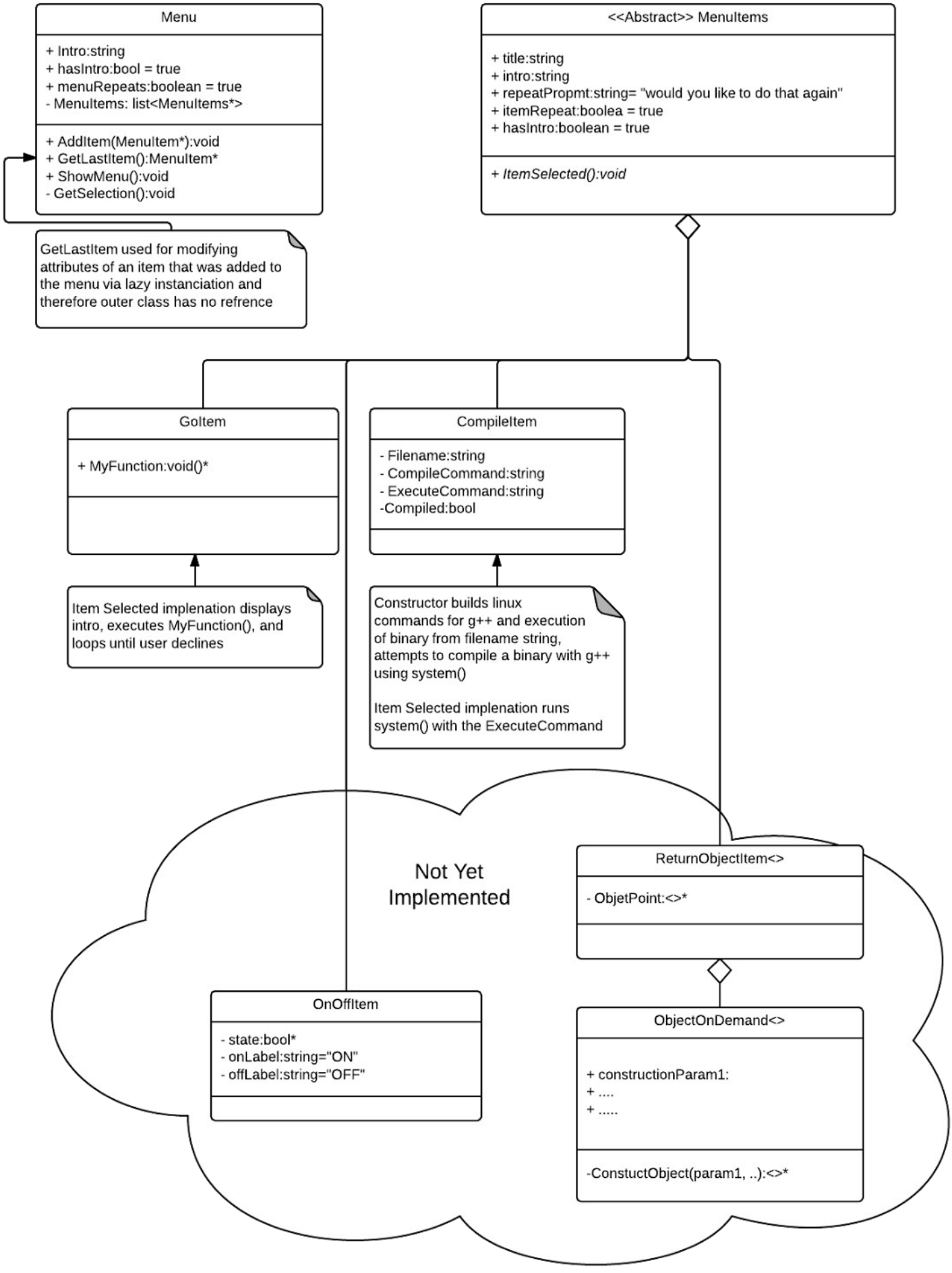
I am currently working on developing more MenuItem subclasses that can be used by my menu subclass. One that can be used for displaying and switching a boolean value, or ON and OFF. It would hold a reference to a boolean value that is a member of the class that created the menu item. And would display its state in the menu item title.

I also want to make a different version of Menu that instead of executing a piece of code, returns an object of some type. This would be a template class so that it would return the type of class that it was instantiated to handle. each menu item would have a pointer to such an object, or the necessary

information to construct such an object.

Attached below is a UML diagram of the Menu and Menu-Item with subclasses. I left out the compile menu object that subclasses Menu. This was a simple convenience class, It does not modify the behavior of the menu but simply makes it easy to construct a menu full of compile menu items by only providing a list of filenames.




Menu
+ Intro:string
+ hasIntro:bool = true
+ menuRepeats:boolean = true
- MenuItems: list<MenuItems*>
+ AddItem(MenuItem*):void
+ GetLastItem():MenuItem*
+ ShowMenu():void
- GetSelection():void
GetLastItem used for modifying attributes of an item that was added to the menu via lazy instanciation and therefore outer class has no refrence
<<Abstract>> MenuItems
+ title:string
+ intro:string
+ repeatPropmt:string= "would you like to do that again"
+ itemRepeat:boolea = true
+ hasIntro:boolean = true
+ ItemSelected():void
GoItem
+ MyFunction:void()*
Item Selected implenation displays intro, executes MyFunction(), and loops until user declines
CompileItem
- Filename:string
- CompileCommand:string
- ExecuteCommand:string
-Compiled:bool
Constructor builds linux commands for g++ and execution of binary from filename string, attempts to compile a binary with g++ using system()
Item Selected implenation runs system() with the ExecuteCommand
Not Yet Implemented
ReturnObjectItem<>
- ObjetPoint:<>*
OnOffItem
- state:bool*
- onLabel:string="ON"
- offLabel:string="OFF"
ObjectOnDemand<>
+ constructionParam1:
+ ....
+ .....
-ConstuctObject(param1, ..):<>*