PART I

1. List at least 8 classes that can be used in a system like Courseworks

* Student
* Class/course
* Professor
* Menu
* File
* Assignments (we could imagine that assignment extends File)
* textEditor
* Administrator

1. Exercise 2.8 from the class textbook
2. university-student – aggregation
3. student-TA – neither (a TA has many more powers and responsibilities. Also, TAs don’t have to be students)
4. student-freshman – inheritance
5. student-professor –aggregation (though the relationship can go both ways: a student has many professors, and a professor has many students)
6. car-door –aggregation (cars have doors)
7. truck-vehicle –inheritance
8. traffic-trafficSign –neither
9. trafficSign-color –aggregation (signs have colors)
10. See end of this file.

PART II

1. Analysis

* User case #1 *User specifies what s/he wants to read*

1. User arrives at main page, by accessing site or by choosing that button from a menu
2. User sees a list of all broad categories of articles
3. User chooses a category or title from the list
4. User clicks and opens the article

**Variation #1**

* 1. In step 3, there exist subcategories of that topic
  2. User sees a list, similar to the one in step 2, until s/he finds an article
* User case #2 *User writes an article*

1. User selects appropriate option from the main-page menu
2. User arrives at a page of warnings, rules, tops of further direction
3. User goes through the necessary instructions
4. User is prompted to sign up or log in
5. User writes and titles his/her article

**Variation #1**

* 1. User does not have a username log in
  2. User is warned his/her IP address will be recorded as a mode of identification
  3. Page records user’s IP address when s/he clicks ‘continue’
  4. User writes article
* User case #2 *User reads an article*

1. User arrives at main menu
2. User inputs desired article or keyword in search box
3. User arrives at desired article

**Variation #1**

* 1. The article user is looking for is not found
  2. Other articles under the same category or related categories are suggested
  3. User chooses one and arrives at article

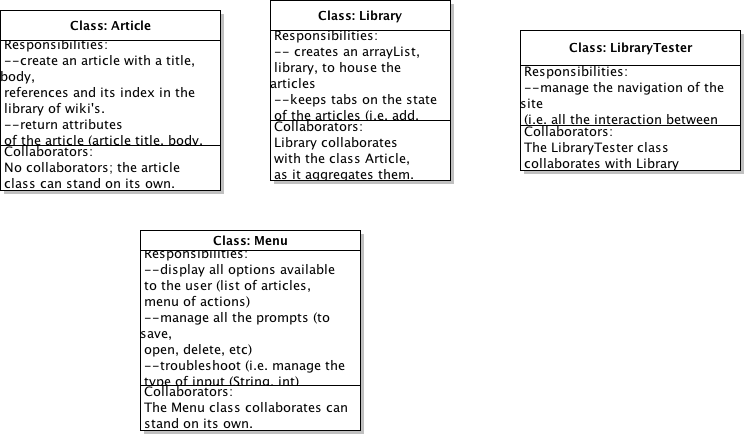
**Variation #2**

* 1. The article or topic foes not exits
  2. User is told the article s/he is looking for does not exist
  3. User Is given the option to create the article

**Variation #3**

* 1. User is unsure what to search
  2. User is also presented with a list of categories
  3. User chooses a category and/or subcategory
  4. User arrives at article

1. **Design - CRCs**



Initially, I had designed the Tester class to include the responsibilities of Menu (mostly because I had always done it this way, and the prompts the programs I have built required were never more than a sentence or two) but decided to split them up here. The user has many options throughout the program, and it cleaned up Tester to not have to deal with them.

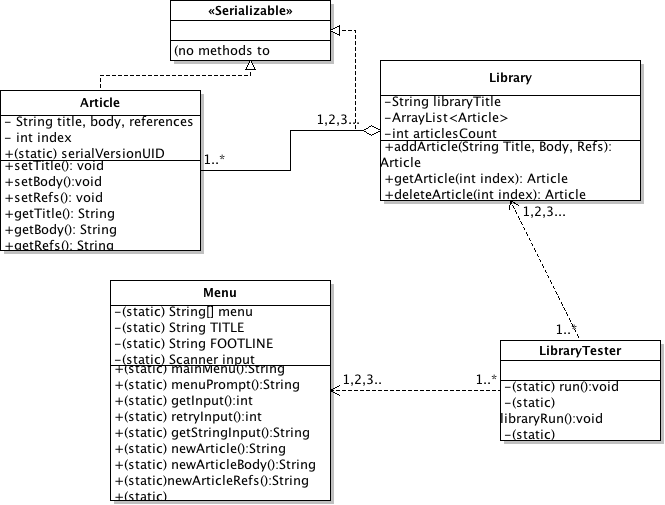
I designed the classes I did by following the “noun method:” I went through the assignment rubric and picked out important nouns and verbs.

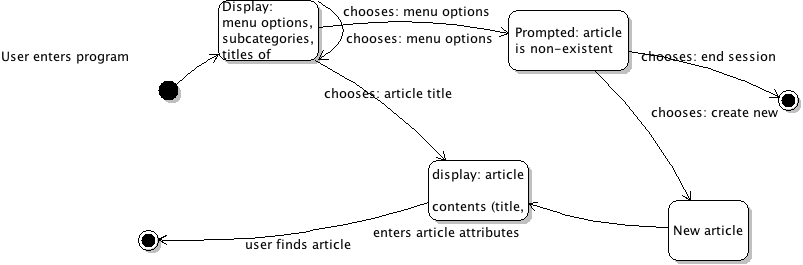
*NB: this class diagram was done before coding*

*NB: I couldn't figure out how to put things on the left (responsibilities) and right (collaborators) of notecards in Violet, so I just decided to split them up like in the above.*

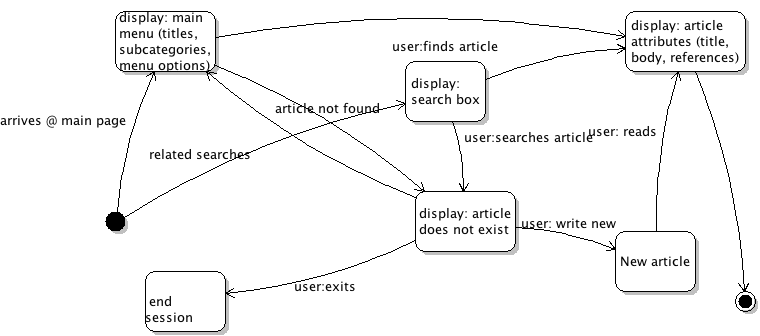
*NB: I didn’t include the Serializable interface as part of my CRCs, in part because it is so mysterious, but also because I did not design it.*

1. **Design UML**

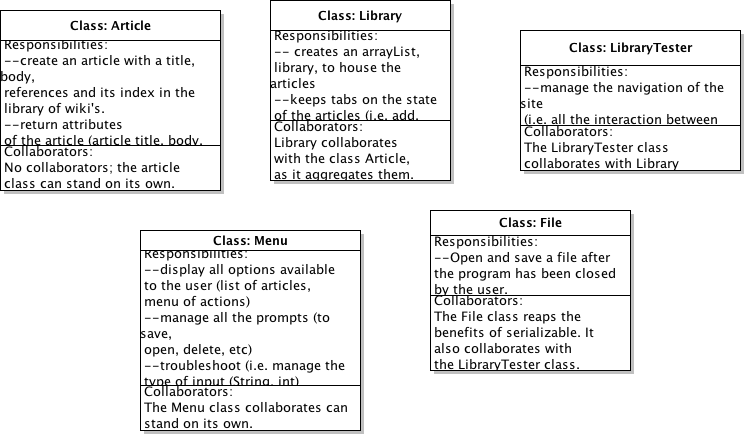
****



*User searches + writes article*



*User wants to read an article*

1. **Changing Design - CRC**

At this point, I decided to split up File from Tester, because it seemed that File had enough responsibilities to merit a different class––in a way, it was a little like the object Article, which could have been clumped in Library.

*NB: this was done before coding. Later, I realized that I could have “File” under LibraryTester and minimize the amount of coupling between my classes, which was already heavy between Article, Library and Library Tester.*

**Part I, cont’d**

3.

It is difficult to form a real opinion about the post, given that most of us––or maybe just me––haven’t been exposed to many languages. From the perspective of someone who is still learning and is still in touch with the world in which programming languages are not common, I can say that Java is, yes, bizarrely object-oriented, but also composed of the right amount of novelty and complexity for a newcomer.

I don’t know what to make of most other languages. All I hear are gripes about C++ (pointers?) while more experienced friends tell me how much “freer” and “easier” Python is. I tried Python two summers ago and foolishly took the straight-forwardness of “print(*x*)” as a given before entering W1004 and learning Java (we have to import Scanners now?). Like the author of the post, I found Java strange and not as clean as the code everyone urged me to write. It does seem rather ridiculous that a noun has to fetch a noun to fetch another noun just so we can borrow the last noun’s methods.

Yet, personally, I don’t think I could have asked to start learning to code with a better language. Java had the right amount of mystery to prevent the thoughtless copying and mechanization of Python code. With Java, I had to find out why I couldn’t just command the console to “print” and instead had to type that pesky “System.out.println().” Coding in Java requires that you’re careful. It’s not good to have parentheses lying around, and it is especially not a good idea to forget your punctuation (I am now able to spot a missing period from miles away!).

I realize that perhaps my loyalties to Java are only sentimental: my first programming language! And is it useful or pretty to design in Java? I can’t really tell, except the fact that the language is still around should say something. Again, from a student who is just learning to code, Java produces the right type of thinking. So it’s not intuitive to fit everything into an object, yeah, but it is not intuitive to most people to code in *any* language either. When I made my first “complex” program, a simulation of Blackjack, I couldn’t get past how odd and magnificent it was that I was designing an intangible “deck” and that in order for that object to “shuffle,” I couldn’t just command it to, but had to fidget with its attributes (maybe switch the top of the deck to have a lower index each time, maybe randomize an index within an ordered deck).

I know that what I see is not necessarily all there is: Java hides its own inner workings like any other language (I was dumbfounded when it struck me that someone had to have written Java in *something that was not Java* to make Java!). But it does in the right measure, just so that if I wanted to dabble in Javascript and marvel at the possibilities of jQuery, I could slip into it much more easily than if I had learned Python.

I can appreciate the author’s point, agree with most of them, even. But I find fault with his underlying principles, the ones that go beyond the computer. Code is supposed to be simple, but coding? I’d think not––especially if by coding “outside the box,” going past our trashcans as we take out the trash, we can get somewhere we wouldn’t have imaged.