Max Siegfried

Project Report

Architecture:

The overall architecture of my project is relatively simple since it is a single class, Bag, which implements a bag as a collection in grace. I’ve added a few helper methods which are duplicated in other files for the purposes of getting them to work, although in reality this class would likely be implemented somewhere else and would not require the additional methods (min and max). The main class acts as an extended interface to dictionary implementing common methods one would want to use in a bag. Most of the methods are self explanatory. Additionally there is a nested class, elementsAndCounts, which returns a list of the bindings of items and the number of times they appear in the bag. The purpose of this class is to allow the user to to iterate over the internal bindings that represent the bag if they need these operations as well.

Challenges:

I faced several challenges while trying to create this class. Although the previous assignment gave me some experience with collections in grace, there are still some elements of the grace syntax that haven’t fully clicked in my head yet. A few new syntax terms I didn’t know came up while reviewing other collection code, such as ‘outer’, but my questions here were answered which helped me move along. For the most part, it just took time to put myself into the mindset of grace which is quite a bit different than language I’m used to writing in.

Additionally, a large component of collections are the iterators. I spent a lot of time thinking about how an iterator could be written in such a way that an internal structure of bindings could be represented as a list, and in the end decided the best way would be to build a list and return that iterator. Additionally the elementsAndCounts class is provided to allow the user to manipulate the bindings directly giving additional options for this particular representation. I had experience like this with a lot of methods, where I was fighting hard to write my own version instead of just finding the class or object that already knew what I wanted it to do. Sometimes, however, it was hard to find the correct object, but this will decrease with more experience with the codebase.

As I’ve been working as an intern as part of the PCEP program, I’ve done a lot of programming as part of a team with a variety of people who know different parts of the codebase well. I think working alone on a larger project like minigrace is difficult because I can only search through so much code so fast, and getting blocked usually means I have no one to turn to. I had hoped that another classmate would be interested in this project to form a team but unfortunately that didn’t happen, so getting stuck on small nuances in the code could end up taking several hours to track down the answer. This happened in instances when I was looking for the correct trait to inherit (although not the new traits not implemented yet) or the correct type to use, i.e. Iterable, lazySequenceOver, Collection, etc. I think the choices I made are made with the intention of being simple and readable.

Additionally, and while this isn’t a particularly interesting area I had a lot of struggles with configuration on my machine. I probably should have given up and tried a different machine early on but the process of making, installing, and compiling on my own Linux box ended up taking a lot of time.

Additional items I’m proud of:

While not the most impressive from a code writing perspective, the most impressive thing I think I did was finding the classes to do the jobs I wanted them to do. You briefly mentioned abstracting the hash table from both dictionary and this class, but I felt that I would still end up wanting an interface for the hash table that mostly looked like the existing dictionary class. As you can see from my code, the interface’s ease of use allowed me to write very short, readable code with little need for additional explanation. This, along with using inheritance to get key methods such as using the inherited do seperatedBy in my string function by providing an iterator method, allowed the code to stay extremely concise.

Throughout the process I used a test driven approach of writing the unit tests and trying to pass them. Most of the unit tests were inspired and/or copied by the existing set and dictionary tests, but they allowed me to think about what I really wanted the bag class to do and therefore solve some of the problems I mentioned earlier, such as how the iterator would work in this representation. I had implemented some TDD before but not in the context of building up a large, multifunction class from scratch. Sticking with this design, even though I often had a quick idea of how to implement a new function, gave me better insight into the implementation as well because I was implementing to solve a problem as opposed to create a thing.

I think the code largely takes a lot of good concepts from POODR, such as designing using inherited (collection) and composed (dictionary) behavior. The bag class is really an extension of the collections library, not a complete rework, and therefore additional methods or features flow very naturally. The tests are additionally designed to be easy to extend, as anyone could plug in a new case with a line or two of code and write a method to solve the problem that is equally as long. Overall, once I had gotten over the challenges which were largely unrelated to the structure of the code (and more related to the syntax), writing the methods was relatively painless and an enjoyable experience to work with.