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.

Additionally I began to sketch out a two way dictionary class inheriting the dictionary class found in newCollections.grace which implements traits. The basic idea is that it will maintain another internal hash table which maps the bindings by value (and reverses the binding pairing). It leads to most methods being pretty simple to create: when adding, add the other binding to valuesInner. When searching, search both internal hash tables. The more interesting ideas come from things like the values / keys list. In my implementation, keys and values will stay as they are in standard dictionary (when writing “at (key) put (value)” means that only the first argument is a key) but the user is free to retrieve (using “at”) using either the key or value. I think this gives the user more flexibility in how they use the collection and avoids them getting iterators back full of duplicate information. Also, it would make the concept of keys and values completely irrelevant (as they would be the same). I also added a type spec describing the interfaces for both the bag and two way dictionary types.

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.

Aside from the additional environmental issues I have had, a problem I ran into was expecting the newCollections.grace module to be fairly functional, but I’ve found that it’s written using many outdated methods that will no longer compile. Given this, I’ll have to be going in a different direction than my original intent of creating a two-way dictionary by using the traits in this module.

I began by factoring out a lot of the methods that wouldn’t compile (i.e. []) and removing instances where the inheritance was undefined (i.e. graceObject ==). However, even once I was able to get all of the modules to compile there appeared to still be a flaw in the newCollections.grace file as my tests would run forever with no output. I tried to get even one simple test to run but the compiler / tests would never finish and I saw no output. I spent a lot of time trying to track down this error but to no success. I decided to try to use collectionsPrelude as I did with the Bag object, however I wanted to take a different approach. I started with an extremely simple inheritance approach on the standard dictionary and tried to run the suite of tests against it:

class twoWayDictionary<K,T> {

inherits cp.dictionary<K,T>

method asString { "a two-way dictionary factory" }

method withAll(initialBindings: Iterable<Binding<K,T>>) -> Dictionary<K,T> {

def result = empty

for (initialBindings) do { b -> result.add(b) }

result

}

class empty -> Dictionary<K,T> {

inherits cp.dictionary.empty

}

}

However, even this has problems due to the way ‘outer’ currently behaves. There are some cryptic comments in the dictionary class regarding this but I’m not sure exactly what they mean. I received the following and similar errors:

NoSuchMethod on line 1751: no method 'outer' in collectionsPrelude.dictionary.empty.bindingsIterator a collectionsPrelude.dictionary.empty.bindingsIterator.

So, having not achieved much in the way of object oriented design because of the various errors, I decided to sketch out what the implementation might look like using inheritance from newCollections.grace without testing (!). I know this is generally bad practice but wanted to get an idea of what it would look like before I could resolve the technical issues

After getting some help with grace, I went back to the refactor of the newCollections module. The problem I ran into here is that when inheriting from an object, method overrides happen at the time of initialization, but member variables are not yet initialized. So when calling the code to initialize a dictionary, which uses the at() put() method, the new at() put() using the twoWayDictionary’s valuesInner is called, but that variable is not defined. Another refactor of the class is required to fix this problem, however even when inheriting from dictionary and including no initial bindings it still attempts to access the ‘valuesInner’ variable which has not been instantiated, so I’m not sure how to initialize the base class using a method I’ve overriden.

Ultimately I didn’t get every dictionary interface test to pass, but I was able to successfully test my methods (two way get). I think this is due to the ‘into’ method on values not working due to a bug somewhere in the enumerable trait. I left the tests as they were because they still will not be complete until those tests pass, but at the moment I could not track down every bug. As you suggested, I shouldn’t have tried to use the out of date code in new collections although most features are working. There is still some refactoring to be done in the new collections but hopefully some of this work was useful in correcting a few of the bugs.

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.

After writing the type spec, I also feel like these interfaces turned out to be rather simple and elegant with easy to understand interfaces. The two way dictionary interface maintains nearly the same interface as dictionary, but allowing for two-way lookup, removal, and information about whether or not the element is contained in it. The bag class also had an elegant interface, maintaining the same interface as set modified to behave as a bag should, and a few additional methods that provide simple, helpful functionality.