Team: Adam Wiemerslage
Alex Killian
Kenneth Wapman

Title: Multifaceted Dictionary

1. Features that were implemented:

User Requirements	
ID	**Requirement**
UR-01	User can add an Entry.
UR-02	User can add a Word Sense to an Entry.
UR-03	User can add a Definition to a Word Sense.
UR-04	User can add a Word Form to a Word Sense.
UR-05	User can add a Part of Speech to a Word Sense.
UR-06	User can add entries in batch (Word/Definition pairs only) from a file.
UR-07	User can update an Entry word spelling.
UR-08	User can update a Word Form spelling.
UR-09	User can update a Definition.
UR-10	User can update a Part of Speech.
UR-11	User can lookup Entries by word.
UR-12	User can lookup Entries by Definition.
UR-13	User can lookup Entries by Part of Speech.
UR-14	User can remove an Entry.
UR-15	User can remove a Word Sense.
UR-16	User can remove a Part of Speech.
UR-17	User can remove a Definition.
UR-18	User can remove a Word Form.

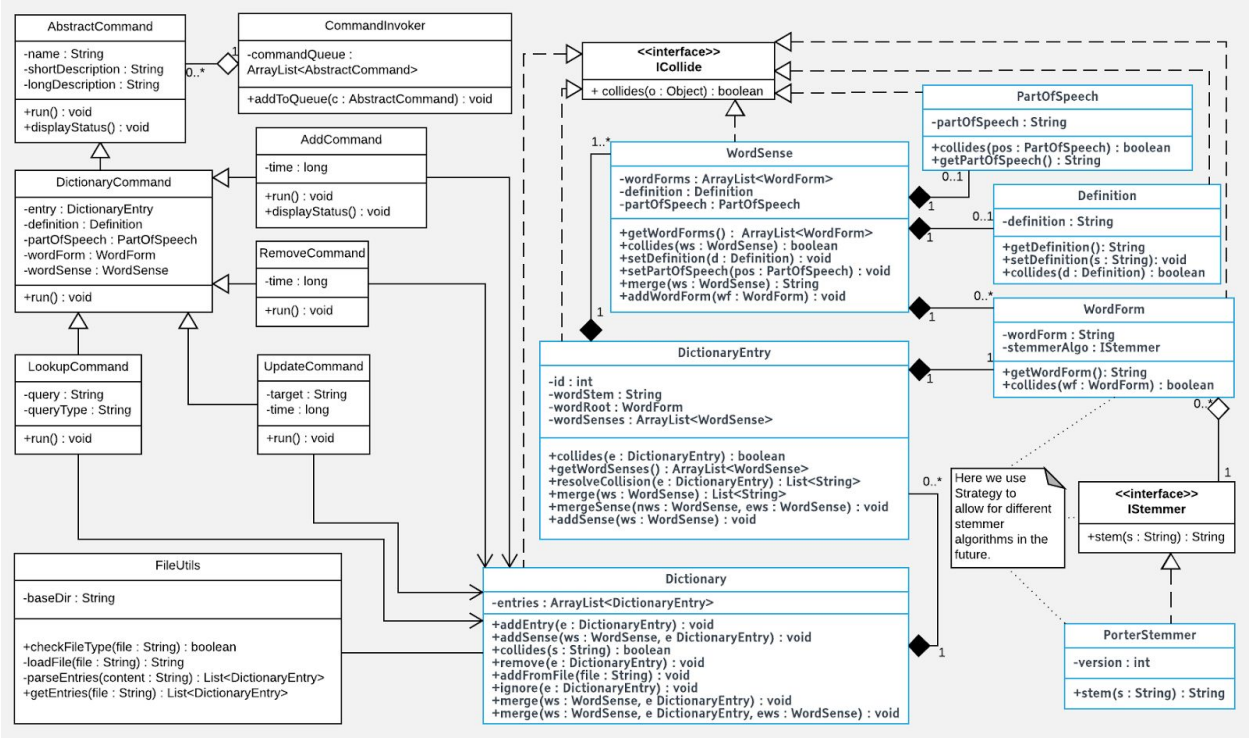
Note: We have a large unit test covering all of the features we implemented.

2. Features were not implemented from Part 2:

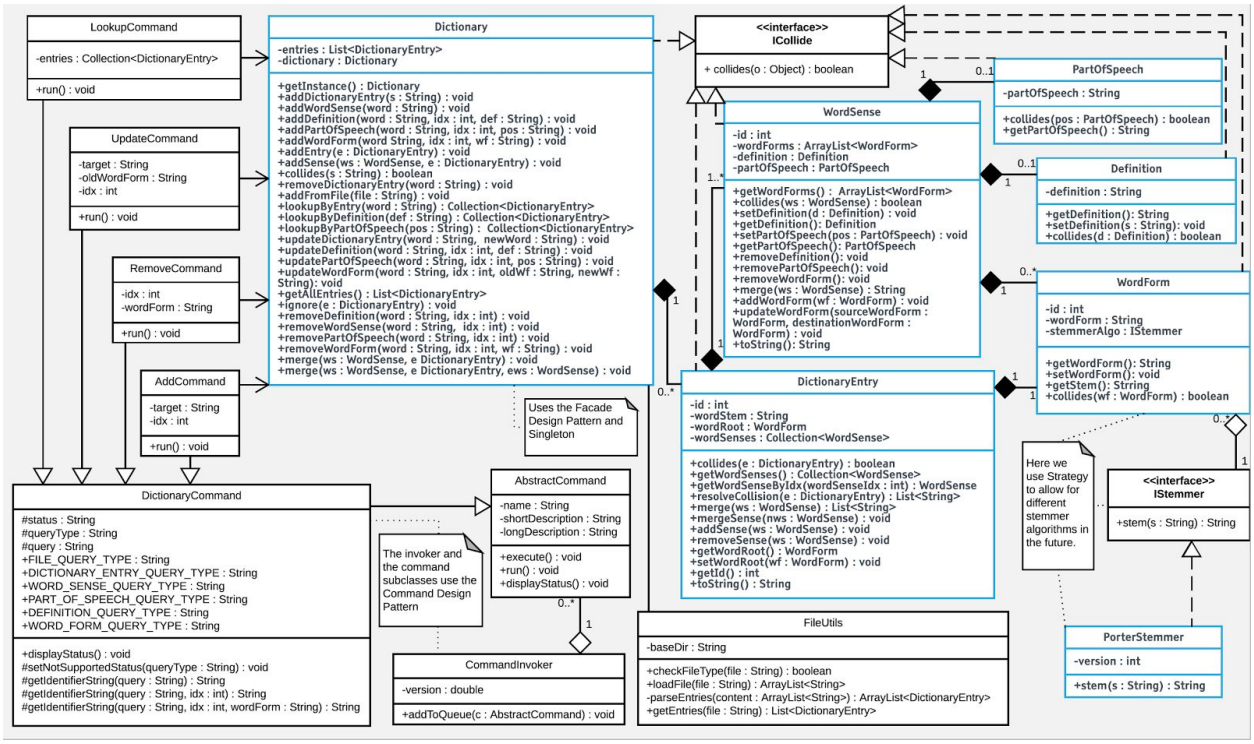
User Requirements	
ID	**Requirement**
UR-19	User can resolve a Collision.

3. Part 2 class diagram and final class diagram. What changed? Why?

Part 2 Diagram:



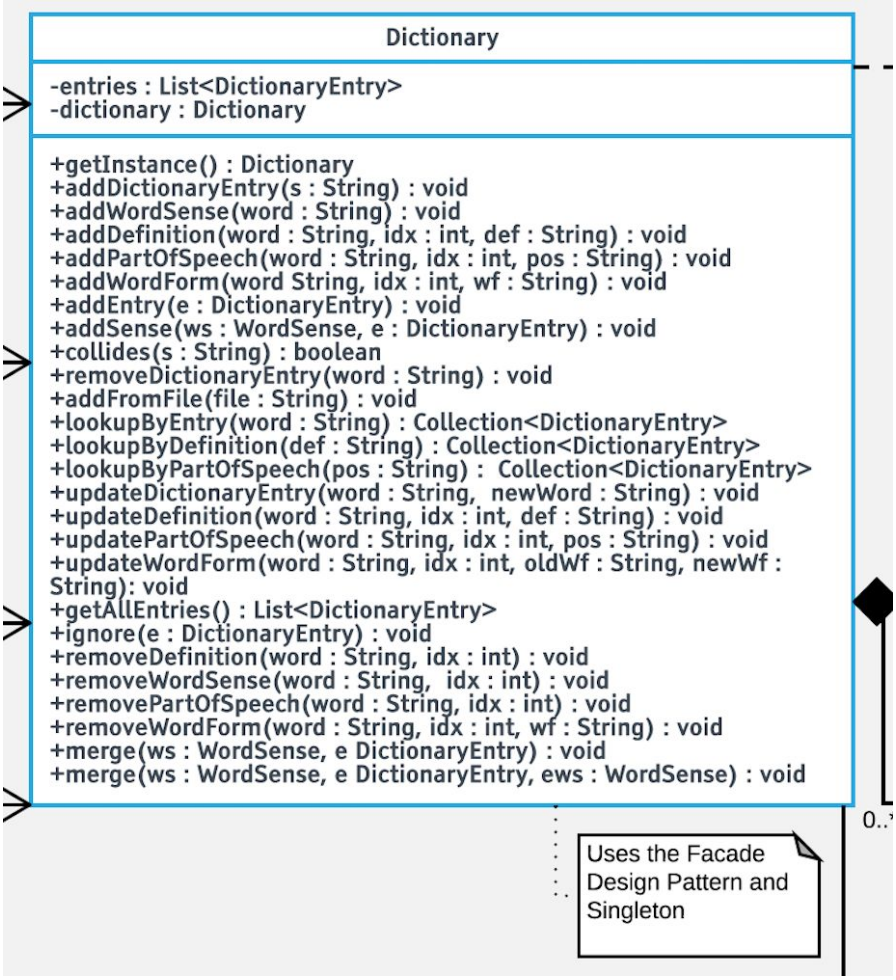
Final Diagram:



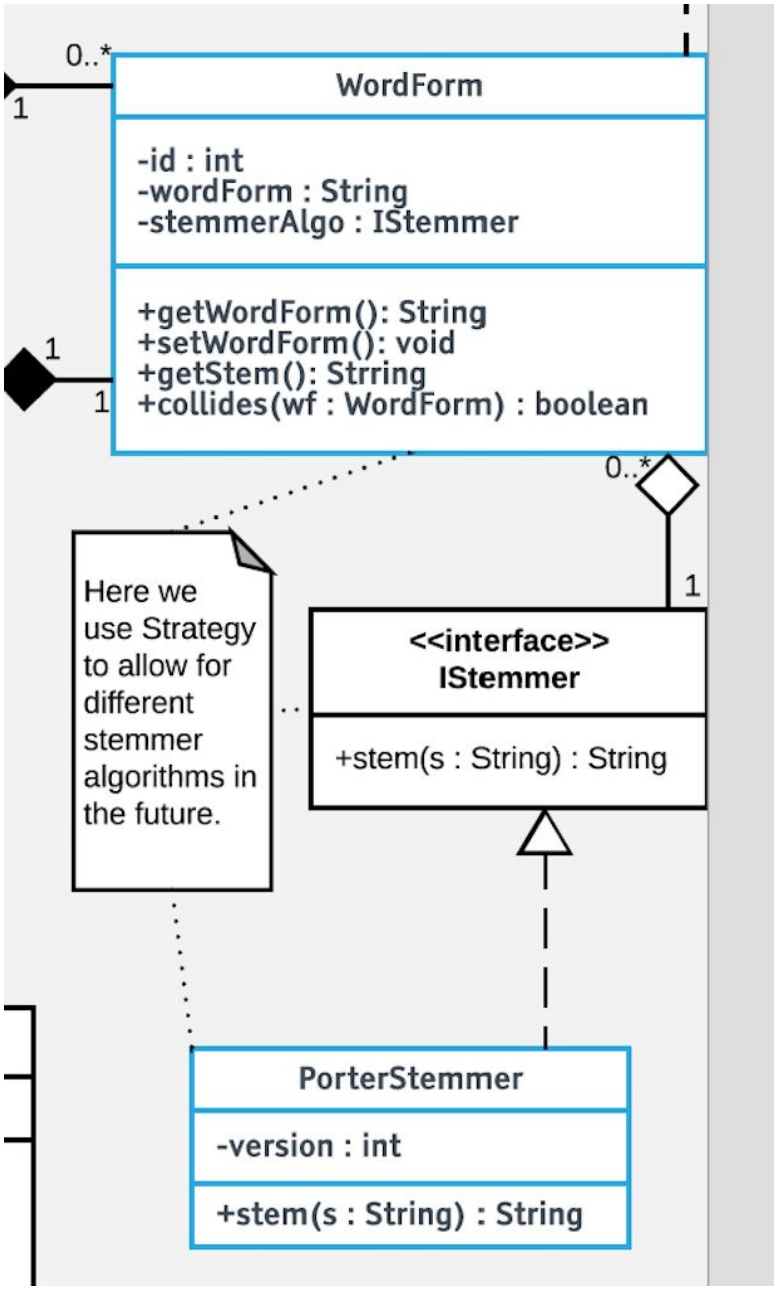
Several methods and properties that were necessary for the user requirements were not in the original class diagram. However, the overall design from part 2 worked well in practice, as all we had to do was add these missing methods. Doing the design upfront meant that we could start creating the classes and filling them in without second guessing what we were adding. It allowed us to get a system up and running without having to answer any critical design questions while we were writing code.

4. Did you make use of any design patterns in the implementation of your final prototype?

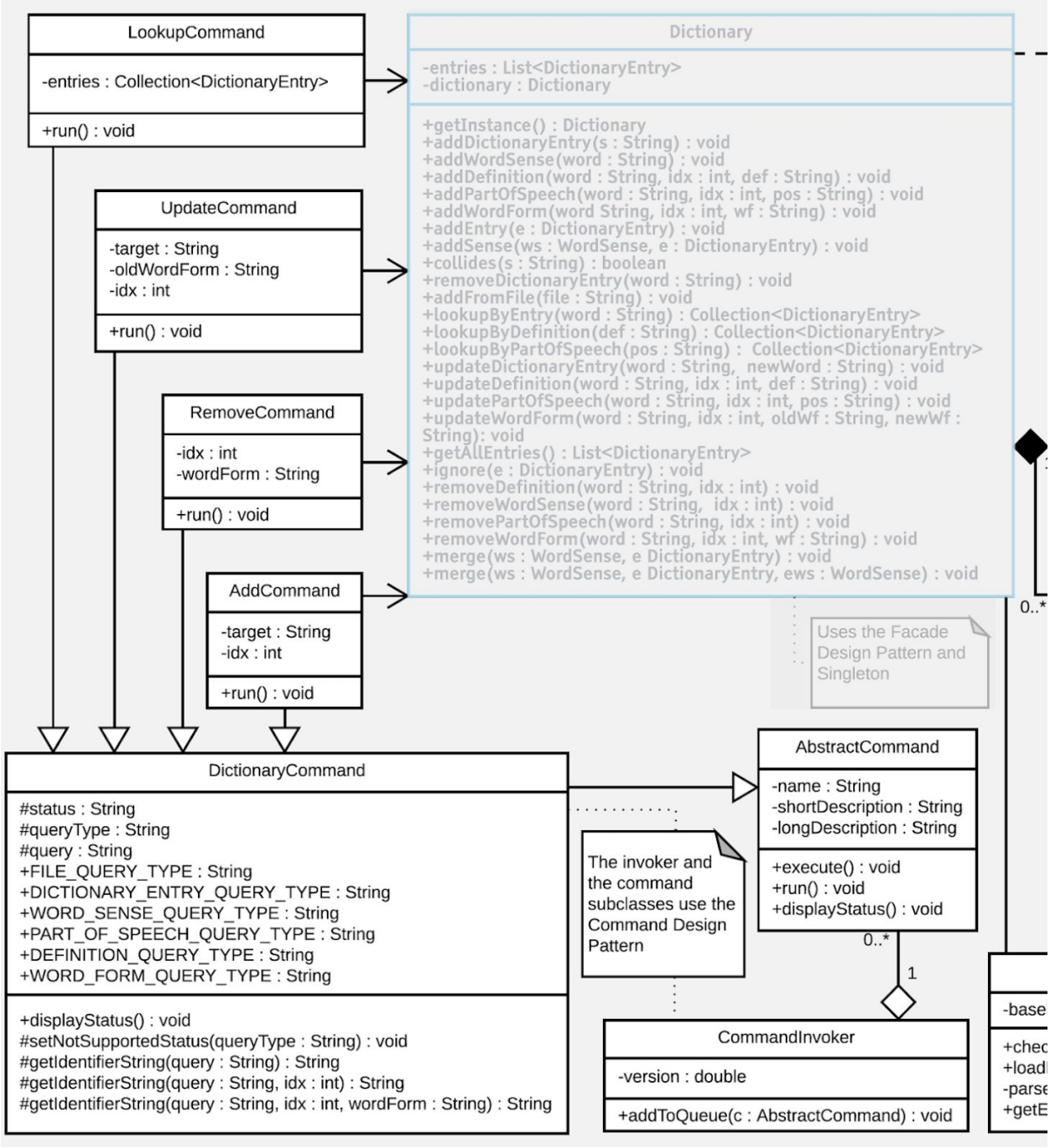
We used the **Facade Controller** pattern for our main dictionary class. This was useful because it allowed us to provide a single access point to the rest of the model. Also, we used the singleton design pattern for the Dictionary, which allowed us to have one common instance and not have to worry about creating more instances.



We used the **Strategy** design pattern in order to make it possible to support different stemming algorithms.



We used the **command** design pattern in order to modularize the different possible commands for interfacing with our dictionary.



Lastly, we had planned on using the flyweight design pattern for reusing the finite set of Part of Speech tags in our system. Given our database it was not, however, necessary to employ this design pattern.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

The process of thinking through all of the use cases early on in the development process was an enlightening exercise. This allowed us to conceptualize our system from the users' point of view before getting involved with the technical details. Completing and refactoring the class diagram before beginning to write code also allowed us to assess which design patterns we were using, or could potentially make use of in implementation. Ultimately, we learned that it is very easy to overlook minor details. However, if your design is good, this problem is inconsequential. This is because a well designed system is easy to add to.