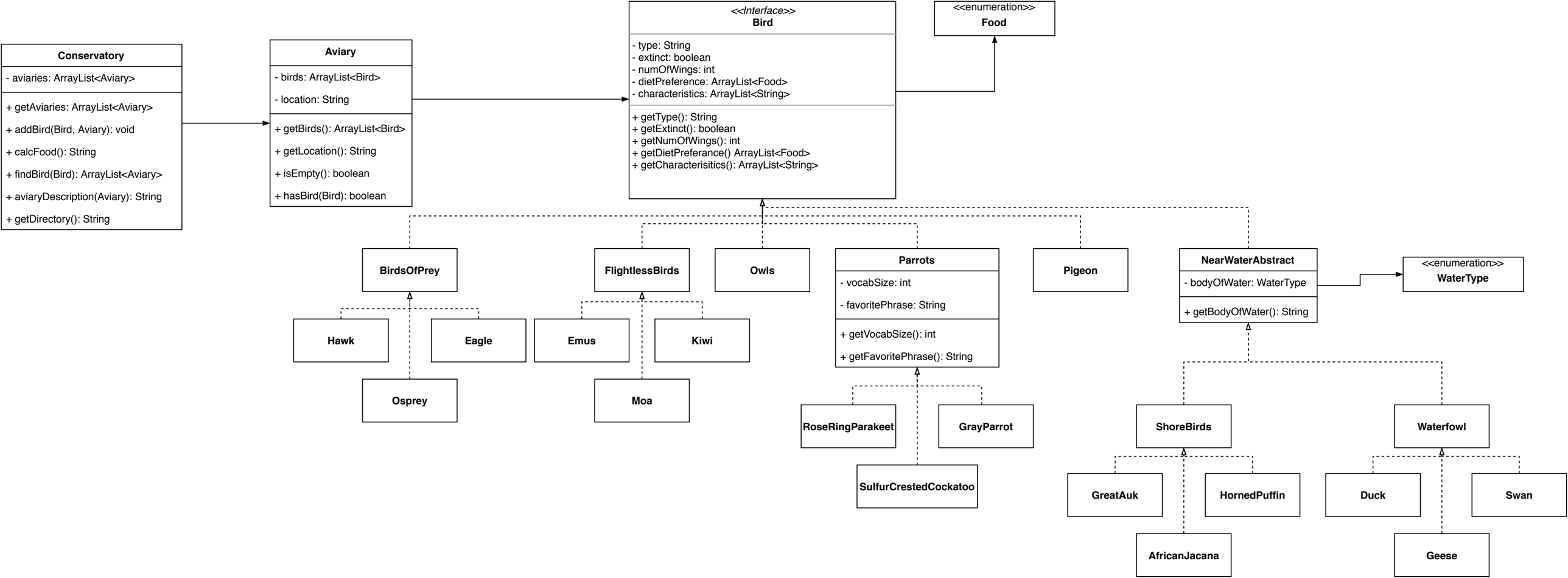
Samuel Steiner

The design for the conservatory, aviary and bird classes are shown in the UML attached to this PDF. A bird interface is used to make sure all birds have the required information. The classifications are used as abstracts to reduce code redundancy. Enumerations are used to maintain finite data for the types of food for diet preference. Each type of bird is a concrete type, there would be types of owls and pigeons but none were listed in the problem description. All fields are private and have ‘getter’ methods which will allow for the user to access them. Overall this will allow for flexibility, if a new bird type is added you just add a new concrete class under the abstract classification, while maintaining that errors due to typos to a minimum, since types can be determined by the concrete class without the need for user input.

Each method would need to be tested for functionality using individual test. This would require multiple test cases with different data to ensure that functionality is working as intended. Testing from the most specific classes to the most generic will ensure that the classes and methods have been implemented correctly following the class structure set by the interface. The aviary and conservatory class can be tested after all the functionality of the Bird interface and its abstract classes and subsequent concrete classes has been test as it relies on those classes for their methods. Testing should also showcase any errors which may be thrown during the usage this is important in the aviary and conservatory class which has several rules listed in the problem description which they must follow. These errors are important to make sure that the classes are invariant.


Conservatory
- aviaries: ArrayList<Aviary>
+ getAviaries: ArrayList<Aviary>
+ addBird(Bird, Aviary): void
+ calcFood(): String
+ findBird(Bird): ArrayList<Aviary>
+ aviaryDescription(Aviary): String
+ getDirectory(): String
Aviary
- birds: ArrayList<Bird>
- location: String
+ getBirds(): ArrayList<Bird>
+ getLocation(): String
+ isEmpty(): boolean
+ hasBird(Bird): boolean
<<Interface>>
Bird
- type: String
- extinct: boolean
- numOfWings: int
- dietPreference: ArrayList<Food>
- characteristics: ArrayList<String>
+ getType(): String
+ getExtinct(): boolean
+ getNumOfWings(): int
+ getDietPreferance() ArrayList<Food>
+ getCharacterisitics(): ArrayList<String>
<<enumeration>>
Food
BirdsOfPrey
Hawk
Eagle
Osprey
FlightlessBirds
Emus
Kiwi
Moa
Owls
Parrots
- vocabSize: int
- favoritePhrase: String
+ getVocabSize(): int
+ getFavoritePhrase(): String
RoseRingParakeet
GrayParrot
SulfurCrestedCockatoo
Pigeon
NearWaterAbstract
- bodyOfWater: WaterType
+ getBodyOfWater(): String
<<enumeration>>
WaterType
ShoreBirds
GreatAuk
HornedPuffin
AfricanJacana
Waterfowl
Duck
Swan
Geese