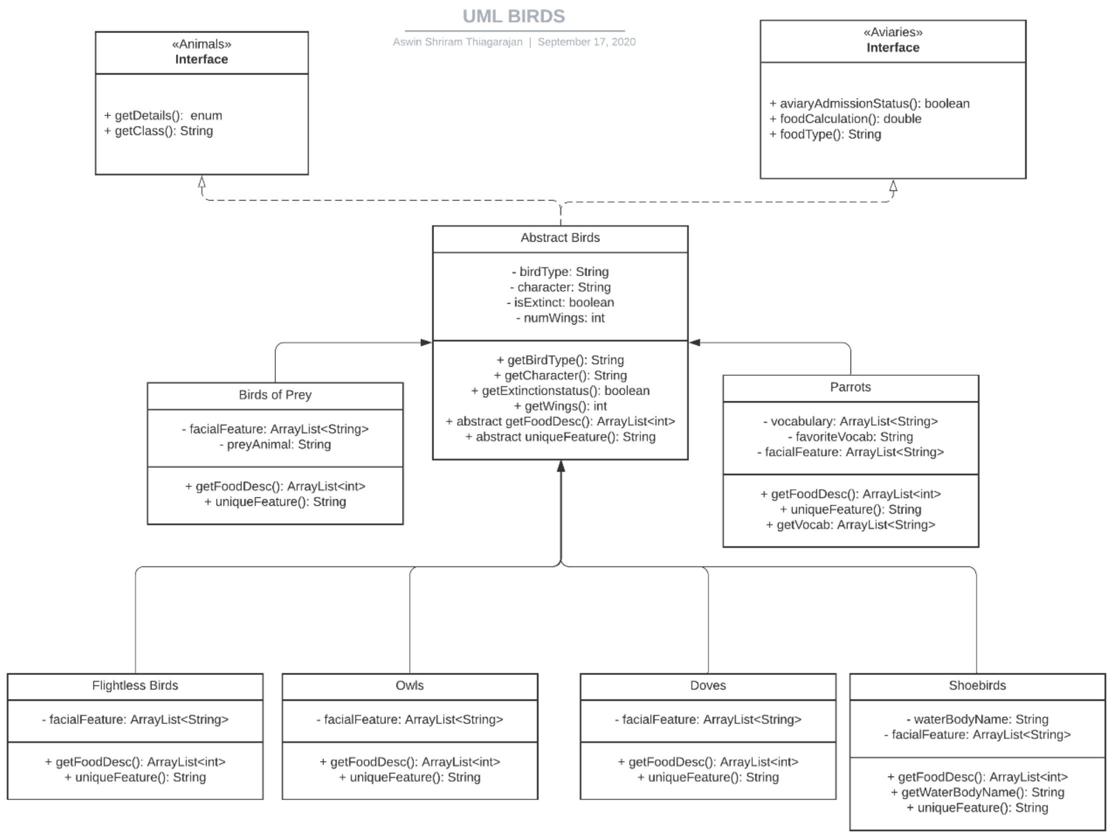
- **Design Explanation:**
  - Created high-level interfaces to offer abstraction and compatibility with the class and their sub-class.
  - Created Abstract class Birds with abstract methods that needs to be overwritten by each sub-class that is inheriting it.
  - Sub-classes are classification of birds that have different properties and characterestics.

- **UML Diagram:**


UML BIRDS
Aswin Shriram Thiagarajan | September 17, 2020
«Animals» Interface
+ getDetails(): enum
+ getClass(): String
«Aviaries» Interface
+ aviaryAdmissionStatus(): boolean
+ foodCalculation(): double
+ foodType(): String
Abstract Birds
- birdType: String
- character: String
- isExtinct: boolean
- numWings: int
+ getBirdType(): String
+ getCharacter(): String
+ getExtinctionstatus(): boolean
+ getWings(): int
+ abstract getFoodDesc(): ArrayList<int>
+ abstract uniqueFeature(): String
Birds of Prey
- facialFeature: ArrayList<String>
- preyAnimal: String
+ getFoodDesc(): ArrayList<int>
+ uniqueFeature(): String
Parrots
- vocabulary: ArrayList<String>
- favoriteVocab: String
- facialFeature: ArrayList<String>
+ getFoodDesc(): ArrayList<int>
+ uniqueFeature(): String
+ getVocab: ArrayList<String>
Flightless Birds
- facialFeature: ArrayList<String>
+ getFoodDesc(): ArrayList<int>
+ uniqueFeature(): String
Owls
- facialFeature: ArrayList<String>
+ getFoodDesc(): ArrayList<int>
+ uniqueFeature(): String
Doves
- facialFeature: ArrayList<String>
+ getFoodDesc(): ArrayList<int>
+ uniqueFeature(): String
Shoebirds
- waterBodyName: String
- facialFeature: ArrayList<String>
+ getFoodDesc(): ArrayList<int>
+ getWaterBodyName(): String
+ uniqueFeature(): String

- **Testing Plan:**
  - Test the fields of the classes via their respective get methods. First instantiate the objects in setup method (@Before), then assert the variables are equal and are of expected data type.

  - For methods that require calculations, check in the test case whether they generate expected results by cross checking against manually calculated values.
  - For example, I would create an object for each subclass (say a bird cockatoo belonging to Parrots class and assert all the outputs of methods like facialFeatures, favoriteVocab, etc.

- **Challenges:**
  - Deciding when to use interface and abstract classes.
  - Various assumptions that makes the above choices dependent of them.