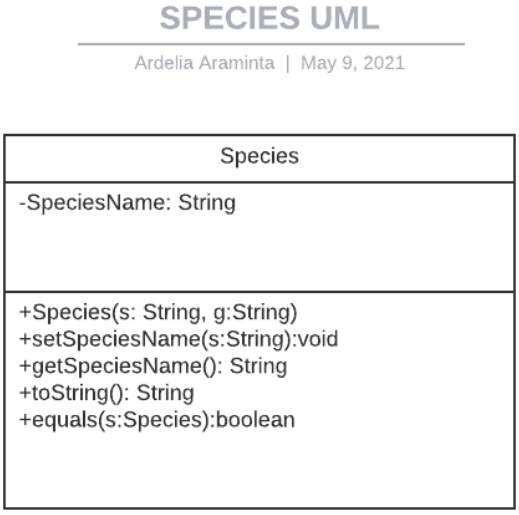
**Forum OOP WEEK 9**

Ardelia Shaula Araminta

2440065163

**Question 1**

a) Genus is the parent class( Superclass) and Species is the child class (Subclass) , so therefore they have parent-child relationship (Inheritance).
b) Specimen has composition relationship with Species mainly because there will be no specimen without Species since entities of specimen are dependent on Species.
c)




d) Inheritance will make the code more efficient and readable since we can use the same functions and codes in the same class since it inherits the properties from the superclass, so the main advantages of this is reusability and readability. On the other hand, composition allows you to reuse code without modelling and is-a association as you do by using inheritance. That allows stronger encapsulation and makes your code easier to maintain.

e) Part I : toString() can be invoked since it is overridden from the parent's class as Inheritance allows us to this in the child class, therefore it doesn't cause error.

Part II: the method is called overriding.

**Question Set 2**

a) Encapsulation is when the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which it is declared.

b) Benefits of Encapsulation:

- It prevents the other classes to access the private fields.
- Encapsulation allows modifying implemented code without breaking other code who have implemented the code.

c) Accessor ( getter) -> getName(), getCage(), getTOA()

d) Instances -> name, cageNumber

e) Github 😊

```java
package OOP;

public class Genus {
    private String gen;
    //constructor
    public Genus(String gen){
        this.gen = gen;
    }
    //accessor
    public String getGenusName() {
        return gen;
    }

    // toString

    @Override
    public String toString() {
        return " Animal Genus: " + gen;
    }
}
```

f) Advantage = Specimen place lower than Species in the hierarchy, the relationship defines with Specimen act as the child (sub-class). Hence, the specimen can access all the data and methods inside its parent class (Species).

- Disadvantage = it is more inflexible as further subclasses customization will be harder because all the methods and data are defined in a more general superclass.

**Question Set 3**

a) Create a private variable with any name but let's make it easier by naming it "marking" with data type of String. In the Specimen constructor, we add another parameter which is 'String marking' and add 'this. marking = marking'. Other than that, we can create a setter and getter method for this.

b) Code below in the Screenhot

**Question set 4**

a) Abstract Data type (ADT) is a type (or class) for objects whose behaviour is defined by a set of value and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation-independent view. The process of providing only the essentials and hiding the details is known as abstraction.

b)

```
    //3b
    public static int countSpecimens(Specimen[] animals, Species s) {
        int count = 0;
        for (Specimen x : animals) {
            if (x.getTOA().getSpeciesName().equals(s.getSpeciesName())) {
                count++;
            }
        }
        return count;
    }

    //4b
    public static LinkedList<Species> makeSpeciesList(LinkedList<Specimen> animals) {
        LinkedList<Species> speciesList = new LinkedList<Species>();
        for (Specimen animal: animals) {
            speciesList.add(animal.getTOA());
        }
        return speciesList;
    }
    //4c
    public static LinkedList<Species> makeSpeciesListUnique(LinkedList<Species> allSpecies) {
        LinkedList<Species> speciesListUnique = new LinkedList<Species>();
        for (Species species: allSpecies) {
            if (!speciesListUnique.contains(species)) {
                speciesListUnique.add(species);
            }
        }
        return speciesListUnique;
    }
}
```