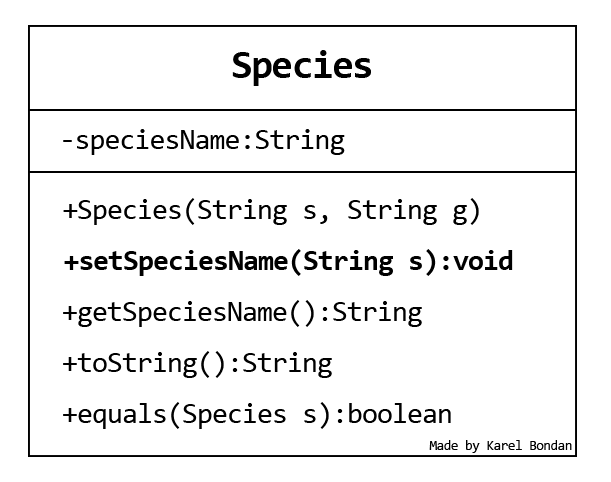
**Question Set 1**

1. Genus is the parent class of the Species class. Genus class inherits its methods and behaviours to Species class, therefore having parent-child relationship. In this case, the relationship between the two classes is **Composition**, because a species is part of a genus, not a genus itself. A composition relationship means that a part of an object will not exist without its other part. In this case, the Species class is impossible to be created without the Genus class.
2. Although in the code snippet there “isn’t any relationship” between the Species and Specimen class (Species class does not inherit its methods and behaviors to the Specimen class), they have a **Composition** relationship. Without the Species class, the Specimen class cease to exist because scientifically too, specimen is part of a species. In this code snippet, the evidence of Species and Specimen class having a composition relationship is in the constructor parameter. Specimen constructor requires a third parameter species with the datatype that is the Species class.
3. UML diagram of the Species class:



1. **1.** By having a Composition relationship between the three classes, necessary information will not be left behind, such as the animals’ genus and species when inputting new data into the system, making the zoo management more efficient.

**2.** By dividing the taxonomy into parts, inputting new data will be more efficient. Say there is already an animal which is a Gray Wolf (or scientifically called as Canis Lupus). When a new Gray Wolf arrives in the zoo, the programmer does not need to initialize a new Species object. The programmer only needs to pass the previously created Species object for the newly arrived Canis Lupus into a new Specimen object.

1. **(i).** The Species class inherits all methods and behaviors from the Genus class. In the code snippet, the Species class’ constructor requires a second String parameter which is g, a possible abbreviation of genus. In the class constructor scope, specifically in the first line, the super() method is called, which will inherit all methods and behaviors from its parent class, the Genus class. The Genus class already implements the toString() method. Therefore, the code will not throw an error although the Specimen class have not implemented the method yet.

**(ii).** Polymorphism; overriding.

**Question Set 2**

1. Encapsulation means wrapping the variables and code acting on the variables as a single unit. This method is used to hide sensitive information stored in variables from user. The attribute to hide the variable is called private. The hidden data can then be accessed and modified by the getter and the setter function.
2. **1.** User cannot directly modify and access the variable whatever they want, giving the class full access control to the variable.

**2.** The fields of a class can be made read-only or write-only.

1. **public** String getSpeciesName(){ **return** speciesName; }
2. **private** String speciesName;

package com.assignment.genus;  
  
public class Genus {  
 private String genusName;  
  
 public Genus(){  
 }  
  
 public Genus(String name){  
 this.genusName = name;  
 }  
  
 public String getGenusName() {  
 return genusName;  
 }  
  
 @Override  
 public String toString() {  
 return "Genus name = " + genusName;  
 }  
}

1. **Advantage:** Specimen objects would inherit all the attributes of the Species object, therefore the methods from the Species class can be accessed directly without having to initialize a new Species object inside the Specimen class.

**Disadvantage:** data in the Species class may not be consistent across the associated Specimens.

**Question Set 3**

1. Add new private variables for the markings (e.g., parentage, birth date, health history and so on); adding getter and setter methods to access and modify them.

public int countSpecimen(Specimen[] animals, Species s) {  
 int specimenCount = 0;  
 for (int i = 0; i < animals.length; i++) {  
 if (s.equals(animals[i].getTOA())) {  
 specimenCount++;  
 }  
 }  
 return specimenCount;  
}

**FUNCTION** listSpecies that accepts parameter of animals list of datatype Specimen

**INITIALIZE** uniqueSpecies of type LinkedList

**FOR** each specimens in the animals list

**INITIALIZE** check of type boolean

**IF** uniqueSpecies is empty **THEN**

**ADD** current specimen to the list

**ELSE**

**FOR** each species in uniqueSpecies

**IF** current species is equal to current specimen **THEN**

**SET** check to **TRUE**

**BREAK**

**END IF**

**END FOR**

**IF** check is **FALSE THEN**

**ADD** current specimen to uniqueSpecies

**END IF**

**END IF**

**END FOR**

**RETURN** uniqueSpecies list

**END FUNCTION**

**Question Set 4**

1. 1. Internal implementation details hidden from user

2. Supports multiple implementations.

3. Constructors are hidden, and no pattern matching is available.

1. Refer to Additional.java and Main.java for the solution (**public** static boolean makeList()).
2. Refer to Additional.java and Main.java for the solution (**public** static LinkedList<Species> makeSpeciesList()).
3. Refer to Additional.java and Main.java for the solution (**public** static LinkedList<Species> makeSpeciesListUnique()).