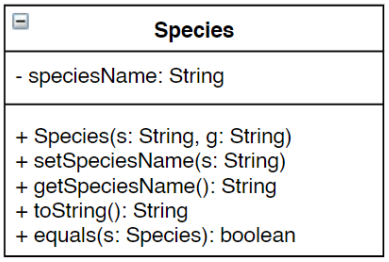
No. 1

a) Species inherits from Genus as Genus is the superclass while Species is the subclass
b) Species is a part of Specimen. It is a variable of Specimen (aggregation)



c)
d) - As the subclass inherits methods from the superclass (i.e. Species and Genus), code can be reused and duplicate code can be avoided
- It can be easier and faster to test code as the methods that are inherited from the superclass do not to need to be tested again for their implementation in the subclasses
e)
    i) The 2 different toString() methods are independently called depending on the class that calls it. As human is an instance of Species, the program / compiler will select the correct toString() method of the class with this case being the Species class' toString() method
    ii) Overriding

No. 2

a) Encapsulation refers to the bundling of data and methods that operate on that specific data. This is used to hide the structure and representation of the actual data within a class, only making them accessible via accessor methods
b) - Security; As the data that is meant to be accessible is only accessible via accessor methods, it ensures that data that is not meant to be accessed cannot be tampered with. It also helps in finding where data is being accessed by simply searching for where the accessor method is being called
- Maintenance; Only the class method needs to be changed if there are any changes to be made to how the data is stored or accessed. Everything else in the code will still use the same accessor method to access the data
c) getName()
d) cageNumber

e)

```java
public class Genus {
    private String genusName;

    public Genus(String genus) {
        setGenusName(genus);
    }

    public String getGenusName() {
        return genusName;
    }

    public void setGenusName(String genus) {
        this.genusName = genus;
    }

    public String toString() {
        return "Genus: " + getGenusName();
    }
}
```

f) - Advantage: Any objects from the Specimen class will inherit all the attributes from the Species class. Methods from the Species class can be reused inside the Specimen class so code does not need to be written again. Methods can also be overwritten inside the Specimen class if needed.
- Disadvantage: Not all methods and data variables from Species, and also Genus as Species is a subclass of Genus, are used in Specimen and thus there would be a potential waste in memory space.

No. 3

a) Add a new private instance variable in the Specimen class named markings. Markings should have its own getter and setter methods. In the toString() method, description for markings should be added and accessed through its setter method. It is optional to add markings as another parameter in the constructor for Specimen.

b)

```java
public int countSpecimens(Specimen[] animals, Species s) {
    int counter = 0;
    for (Specimen animal : animals) {
        if (s.getSpeciesName().equals(animal.getTOA().getSpeciesName())) {
            counter++;
        }
    }
    return counter;
}
```

c)

```java
public String[] listSpecies(Specimen[] animals) {
    LinkedList<String> speciesll = new LinkedList<>();
    boolean found;

    for (Specimen animal : animals) {
        found = false;

        for (String s : speciesll) {
            if (s.equals(animal.getTOA().getSpeciesName())) {
                found = true;
                break;
            }
        }

        if (!found) {
            speciesll.add(animal.getTOA().getSpeciesName());
        }
    }

    return (String[]) speciesll.toArray();
}
```

No. 4

a) - Has a set of values and methods
- Inner workings are hidden from the user
- Operated by the user through a set of operations (interface)

b)

```java
public LinkedList makeList (Specimen[] animals) {
    LinkedList specimenll = new LinkedList<>();

    for (Specimen animal : animals) {
        specimenll.add(animal);
    }

    return specimenll;
}
```

c)

```java
public LinkedList makeSpeciesList(LinkedList animals) {
    LinkedList<Species> speciesll = new LinkedList<>();

    for (Specimen animal : (LinkedList<Specimen>) animals) {
        speciesll.add(animal.getTOA());
    }

    return speciesll;
}
```

d)

```java
public LinkedList makeSpeciesListUnique(LinkedList allSpecies) {
    LinkedList<Species> uniquell = new LinkedList<>();
    boolean found;

    for (Species species : (LinkedList<Species>) allSpecies) {
        found = false;

        for (Species unique : uniquell) {
            if (unique.getSpeciesName().equals(species.getSpeciesName())) {
                found = true;
                break;
            }
        }

        if (!found) {
            uniquell.add(species);
        }
    }

    return uniquell;
}
```