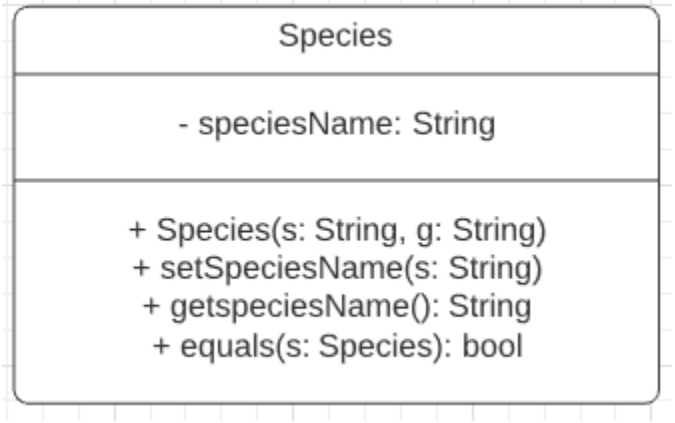
Number 1

a. Genus is the superclass and Species is the subclass. Species inherits the Genus class.
b. Specimens have a Species. It's a composition class
c. 

d. Species inherits all fields and methods from Genus so the code can be reused. An instance of the Species class can be manipulated due to the composition relationship between Specimen and Species.
e. i. This is because the toString() method in the Species class overrides the one in Genus class. Human is an instance of Species so it will use the toString method that is defined in the Species class.
   ii. Overriding

Number 2

a. Encapsulation refers to the bundling of data with the methods that operate on that data. It is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them.
b. It simplifies the maintenance of the application and protects an object from unwanted access by clients.
c. setName()
d. cageNumber

e.

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

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

    public String getGenusName() {
        return this.genusName;
    }

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

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

f. One advantage is that specimen class inherits all methods and instances from the species class so there is no need to rewrite the code. The disadvantage is that it is also a subclass of genus, having too much superclasses might cause memory wastage as data members in base class are left unused.

Number 3

a. A new private string instance variable will be needed to be added. Name it markings. Then, add another parameter to the specimen constructor, String m. add setter and getter for the markings, and make sure to use the setMarkings function in the constructor. Then, edit the toString function to add the description using markings variable.

b.

```
    public int countSpecimen(Specimen[] animals, Species s){
        int count = 0;
        for (Specimen animal : animals) {
            Species toa = animal.getTOA();
            if (s.getSpeciesName() == toa.getSpeciesName()){
                count++;
            }
        }
        return count;
    }
```

c.

```
public class QuestionOne {
    public String [] listSpecies( Specimen[] animals ){

        LinkedList<String> speciesList = new LinkedList<String>();

        for (Specimen animal: animals){

            String species = animal.getTOA().getSpeciesName();
            boolean find = false;

            for (String sp : speciesList) {
                if (sp.equals(species)) {
                    find = true;
                    break;
                }
            }

            if (!find){
                speciesList.add(species);
            }
        }
    }
}
```

Number 4

a. It exports a type. It exports a set of operations. The set is called an interface. Operations of the interface are the one and only access mechanism to the type's data structure.

b.

```
public class QuestionOne {
    public LinkedList makeList( Specimen[] animals )
    {
        LinkedList<Specimen> ll = new LinkedList<Specimen>();

        for (Specimen sp: animals){
            ll.add(sp);
        }

        return ll;
    }
}
```

c.

```
public LinkedList makeSpeciesList( LinkedList animals ){
    LinkedList<String> ll = new LinkedList<String>();
    for (Specimen animal: animals){
        ll.add(animal.getTOA().getSpeciesName());
    }
}
```

d.

```
public LinkedList makeSpeciesListUnique( LinkedList allSpecies ){
    LinkedList<Species> ll = new LinkedList<Species>();
    for (Species species: allSpecies){
        boolean found = false;
        for (Species sp: ll){
            if (sp.getSpeciesName().equals(species.getSpeciesName())){
                found = true;
                break;
            }
        }
        if (!found){
            ll.add(species);
        }
    }
    return ll;
}
```