**Question Set 1**

1. Species is a child class of Genus.
2. Specimen contains a Species object.

|  |
| --- |
| **Genus** |
|  |
|  |

|  |
| --- |
| **Species** |
| * speciesName : String |
| * Species( s : String, g : String) * setSpeciesName( s: String ) * getSpeciesName( ) : String * toString( ) : String * equals ( Species s ) : boolean |

1. Species inheriting the attributes of Genus allows for the programming team to save time in creating the Species class since no repetition is involved in doing so.
2. Since the constructer methods for both Specimen and Species are different, it would be unwise to make Specimen a child class of Species.
4. Even though toString() is defined both in Genus and Species, there is no confusion since only one of those will be executed. In this case, this toString() method will use the one defined in the Speces class.
5. Overriding

**Question Set 2**

1. Encapsulation describes the concept of hiding the internal state or representation of the program from users.
3. It disallows users to directly access variables or methods they should not have access to.
4. It becomes easier to change certain elements of an encapsulated code without influencing the other elements.
5. getName()
6. cageNumber
7. *Check Genus.java*
9. Advantage: Some variables and setters such as those for name can be reused.
10. Disadvantage: Because both classes have different constructors, implementing inheritance might be inconvenient.

**Question Set 3**

1. A new instance variable of type string would be needed along with a setter and getter method.

public int countSpecimens(Specimen[] animals, Species s){

   int matches = 0;

   for(int i = 0; i < animals.length; i++){

        if(animals[i].getTOA().equals(s)){

            matches++;

        }

    }

    return matches;

}



FUNCTION listSpecies( Specimen[] animals){

Specimen[] species;

FOR index from 0 -> length(animals){

Add animals[i].getTOA() to species;

}

RETURN species;

**Question Set 4**

1. The behavior of ADTs defined by a set of values and a set of operations. ADTs only define what operations are to be performed not how the operations itself will be implemented and thus, they are considered to be “abstract”.

LinkedList makeList(Specimen[] animals){

    LinkedList<Specimen> resultList = new LinkedList();

    for(int i=0; i < animals.length; i++){

        resultList.add(animals[i]);

    }

    return resultList;

}



LinkedList makeSpeciesList(LinkedList animals){

    LinkedList<Species> resultList = new LinkedList<>();

    for(int i = 0; i < animals.size() < i++){

        resultList.add(animals.get(i).getTOA());

    }

    return resultList;

}



LinkedList makeSpeciesListUnique(LinkedList allSpecies){

    LinkedList<Species> resultList = new LinkedList<>();

    for(int i=0; i<allSpecies.size();i++){

        if(!resultList.contains(allSpecies.get(i))){

            resultList.add(allSpecies.get(i));

        }

    }

    return resultList();

}