Question Set 1:

A. Inheritance, as we can see from the picture that the species inherits or expands from Genus with Genus being the superclass and species being the subclass.

B. Aggregation, as we can see from the picture that they create a new functionality by combining them into a new class.

C. Species

-SpeciesName:String

+Species(s:String, g:String)

+setSpeciesName(s:String):void

+getSpeciesName():String

+toString():String

+equals(s:Species):Boolean

D. Reusability and Readability: We don’t need to write the same code again in child class. This makes the code much easier and shorter to read while being easier to reuse.

Efficiency: It is much more efficient to use inheritance while writing a code this will result in an increase of speed and a shorter code.

E. 1. Its because species extends from genus so species can call the method from genus while having an override of it’s own

2. Polymorphism

Question Set 2:

1. Encapsulation (Or Data Hiding) in Java is the inclusion of all methods and variables needed for a Java object to function, contained within the object itself.
2. Security (Data Hiding): it can provide the programmer to hide the inner classes and the user to give access only to the desired codes.

Code Flexebility: It can help us to make a flexible code which is easy to change and maintain.

1. Getter method: getName(), getCage(), getTOA()
2. Name, Cage Number, Toa
3. **public** **class** Genus {

**private** String GenusName;

**public** Genus(String g){

**this**.GenusName = g;

}

**public** String getGenusName(){

**return** **this**.GenusName;

}

**public** **void** setGenusName(String g){

**this**.GenusName = GenusName;

}

}

1. Advantage (Reuse Of Code): A subclass can reuse methods that already exists in the superclass.

Disadvantage (Breaks Encapsulation): Subclass needs to be changed if the implementation of the superclass changes.

Question Set 3

A; Add a getter and setter method for marking and also add markings to the override (toString() method)

**public** **class** Specimen

{

**private** String name;

**private** **int** cageNumber;

**private** Species toa; // Type Of Animal

**private** String markings;

**public** Specimen( String a, **int** c, Species s)

{

setName(a);

setCage(c);

setTOA(s);

**this**.markings = markings;

}

**public** **void** setName(String a){

name = a;

}

**public** **void** setCage(**int** c){

cageNumber = c;

}

**public** **void** setTOA(Species s){

toa = s;

}

**public** String getName(){

**return** name;

}

**public** **int** getCage(){

**return** cageNumber;

}

**public** Species getTOA(){

**return** toa;

}

**public** String getMarkings() {

**return** markings;

}

**public** String toString()

{

**return** name + " is a " + toa + " in cage " + cageNumber + " with the marking of " + **markings**;

}

B: **int** countSpeciments (Specimen[] animals, Species s)

{

**int** count = 0;

**for** (Specimen animal : animals)

{

**if**(animal.equals(s))

{

num++

}

}

**return** num;

}

}

C: function listSpecies(animals : Specimen[])

Question Set 4:

1. ADT(Abstract Data Type) is a type of data whose behaviour is defined by a set of operations and values.
2. **public** LinkedList<Specimen> makeList(Specimen[] animals)

{

LinkedList<Specimen> list = **new** LinkedList<Specimen>();

**for** (Specimen animal : animals)

{

list.add(animal);

}

**return** list;

}

1. **public** LinkedList<Species> makeSpeciesList(LinkedList<Specimen> animals)

{

LinkedList<Species> list = **new** LinkedList<Species>();

**for** (Specimen animal : animals)

{

list.add(animal.getTOA());

}

**return** list;

}

1. **public** LinkedList<Species> makeSpeciesListUnique(LinkedList allSpecies)

{

LinkedList<Species> list = **new** LinkedList<Species>();

**for** (Species animal : allSpecies)

{

**boolean** unique = **true**;

**for** (Species s : list)

{

**if** (s.equals(animal))

{

unique = **false**;

**break**;

}

}

**if** (unique)

{

list.add(animal);

}

}

**return** list;

}