1. A. Genus is the parent class of species class

B. Specimen instantiates an instance of the class species as an instance and variable

C. 

D. (1) They can utilize any fields or methods inherited from the genus class without having to rewrite code specific for the species class

(2) They can represent a species as an object in the specimen class and gain access to the public methods provided by the species class

E. (1) Because toString() method in the species class overrides the same method in genus class

(2) Overriding

2. A.Encapsulation means data hiding so certain fields and methods cannot be accessed or used by other class

B. (1) You can prevent other classes from directly accesing and changing certain variables in the class so as to not disrupt the flow of the data in that class

(2) You can Provide input validation or do other operations before the variable updated

C. getTOA()

D. Toa

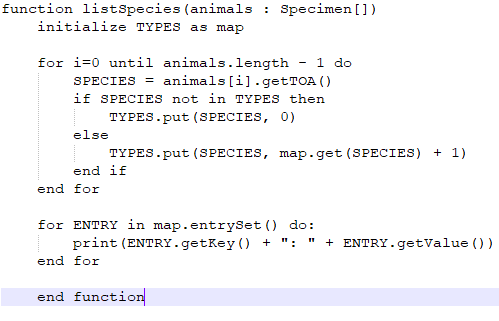
E. package assignment.java;  
  
public class assignment {  
 private String genusName;  
 public genus(String g){  
 this.genusName = g;  
 }  
 public String getGenusName()  
 {  
 return this.genusName;  
  
 }  
 @Override  
 public String toString()  
 {  
 return "genus" + this.getGenusName();  
 }  
}

F. Advantage : The Specimen class will have direct access and inherit any of the public field and methods from the specimen class

Disadvantage : It may prove to become quiet overwhelming to inherit all the fields and methods from the species class since the specimen class doesn’t seem to have the purpose of being a type of species

3. A. A private field called markings would need to be added to the species class mutator and accessor method would need to be added as well in order to get data from the variable and set value to it the toString() method can also be updated to display the value of markings

B.  
  
public int countSpecimens(Specimen[] animals, species s) {  
 int num = 0;  
 for (Specimen animal : animals) {  
 if (animal.equals(s)){  
 num++;  
 }  
 }  
 return num;  
 }

C. 

4. A. Abstract data types have their behavior defined by values and operations but the implementation isn’t explicitly defined

B. public LinkedList<Specimen> makeList(Specimen[] animals) {  
 LinkedList<Specimen> list = new LinkedList<Specimen>();  
 for (Specimen animal : animals){  
 list.add(animal);  
 }  
 return list;  
 }

C. public LinkedList<Species> makeSpeciesList(LinkedList[Specimen] animals) {  
 LinkedList<Species> list = new LinkedList<Species>();  
 for (Specimen animal : animals){  
 list.add(animal.getTOA());  
 }  
 return list;  
 }

D. public LinkedList<Species> makeSpeciesListUnique(LinkedList<Species> allSpecies) {  
 LinkedList<Species> list = new LinkedList<Species>();  
 for (Species animal : allSpecies){  
 boolean unique = true;  
 for (Species a : list){  
 if (a.equals(animal)){  
 unique=false;  
 break;  
 }  
 }  
 }  
 if(unique){  
 list.add(animal);  
 }  
 return list;  
 }