Question set 1

(a) The **Species** class inherits from the **Genus** class

(b) The Specimen class has a member variable that is a Species object; [1 mark]

(c)

|  |
| --- |
| Species |
| String : speciesName |
| setSpeciesName (String s)  String getSpeciesName()  String toString()  Boolean equals(Species s) |

(d) since all function works from the parent class that is inherited, it is not necessary to duplicate the code to reproduce that function in the new class and function that inherited from a parent class will not need to be tested again in the new class

(e) (i) The complier selects the correct method

(ii) Polymorphism or overriding

Question set 2

(a)Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

(b) The ease of testing, putting all structure of data in one class. Other classes which make use of that data can easily test by providing them with data values before the real data class is available for testing

The ease of maintenance. If there are changes with the data or how it stored and accessed internally, only one class needs to be changed to accommodate that, others will continue to use the public accessor methods and will not need to be changed.

(c) getName()

(d) cageNumber

(e) public class Genus

{ private String genusName;

public Genus(String g) {

setGenusName(g); }

public void setGenusName( String g ) { genusName = g; }

public String getGenusName() { return genusName; }

public String toString() {

return "Genus: " + genusName; } }

(f) The Specimen objects will inherit all the attributes of the Species object, this will help to allow the code in the Specimen object to access its methods directly rather than having to invoke them on a Species object using dot-notation. While the disadvantage is that not all methods and variables are consistent between all specimen, it means that changes made to the methods or variable in the Species class may not be applied to all specimens.

Question set 3

(a) Characteristic of an individual animal, which means the description should be an instance variable(s) within the Specimen class, there should be accessor (get/set) methods, the toString()

(b) void countSpecimens( Specimen[] animals, Species s )

{

int sCount = 0;

int i;

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

{

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

sCount++; }

System.out.println( sCount );

}

(c) loop BEAST from 0 to the number of elements in ANIMALS

NEW\_SPECIES = true

UNIQUE.resetNext()

while UNIQUE.hasNext() {

if (UNIQUE.getNext() = species of ANIMALS[BEAST]){

NEW\_SPECIES = false end if end loop if NEW\_SPECIES }

Else if (UNIQUE.addItem( species of ANIMALS[BEAST] ) {

Break}

UNIQUE.resetNext()

while UNIQUE.hasNext() {

SPECIES = UNIQUE.getNext()

System.out.println(SPECIES.toString() )}

Question set 4

1. There is no implementation details that are known
2. Linkedlist MakeList( Specimen [] animals)
3. {

Linkedlist llist = new Linkedlist();

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

{ llist.addHead( animals[i]);}

return llist;

}

1. public void MakeSpeciesListUnique ( Linkedlist allSpecies)

{

Linkedlist uniqueSpecies new Linkedlist();

Boolean founType;

Species foundSpecies;

Species type = (Species) allSpecies.getHead();

while (type != null )

{ foundType = false; f

oundSpecies = (Species) uniqueSpecies.getHead();

while (foundSpecies != null)

{ foundType = false;

foundSpecies = (Species) uniqueSpecies.getHead();

while (foundSpecies != null)

{ if (foundSpecies == type) foundType = true;

foundSpecies = (Species) uniqueSpecies.getNext(); }

if ( !foundType ) uniqueSpecies.addHead( type );

type = (Species) allSpecies.getNext(); }

allSpecies = uniqueSpecies; }