**AssignAssignment 2 – Inheritance**

**Question Set 1**

1. Species extends Genus (“is a” relationship)
2. Specimen class has a member variable that is of the Species object (“has a” relationship)

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

1. They show the actual relationships between each tier in the real world, and using inheritance can prevent the programmers from repeating the codes when writing for the objects.

(i) Because the compiler was able to run the correct method as the two methods are both related to the Species Class.

(ii) Overriding

**Question Set 2**

1. Encapsulation is the mechanism of wrapping the variables and methods together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can only be accessed through the methods of their current class.
2. Benefits of Encapsulation:
   1. protects an object from unwanted access
   2. Increases usability
3. Accessor method: **public int** getCage()
4. Instance Variable: **private int** cageNumber
5. Code:

public class Genus

{

private String genus;

public Genus(String genus)

{

this.genus = genus;

}

public String getGenus()

{

return genus;

}

public void setGenus(String genus)

{

this.genusName = genus;

}

@Override

public String toString()

{

return "Genus{" + "genus='" + genus + '\'' + '}';

}

}

1. Advantage: all the methods found in the Species class will be inherited by the Specimen class, so the programmers will not need to repeat the same codes from both classes.

Disadvantage: some methods in the Species class will not conform to the Specimen class, but the Specimen class still needs to carry those methods.

**Question Set 3**

1. Changes to be made:
   1. We have to make an instance variable within the Specimen class for the description of each individual’s markings.
   2. We have to create getter and setter methods for the markings.
2. Code:

public 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 );

}

**Question Set 4**

1. Features of ADT:

a. Interface

b. Memory

c. Functions/ Methods (Public and Private)

1. Code:

LinkedList makeList( Specimen [] animals )

{

LinkedList specimenList = new LinkedList();

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

{

specimenList.addHead( animals[i] );

}

return specimenList;

}

1. Code:

public LinkedList makeSpeciesList( LinkedList animals )

{

LinkedList sList = new LinkedList();

Specimen indivSpecimen = (Specimen) animals.getHead();

while (indivSpecimen != null)

{

sList.addHead( indivSpecimen.getTOA() );

indivSpecimen = (Specimen) animals.getNext();

}

return sList;

}

1. Code:

public void makeSpeciesListUnique( LinkedList allSpecies )

{

boolean found;

LinkedList unique = new LinkedList();

Species current;

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

while (speciesType != null )

{

found = false;

current = (Species) unique.getHead();

while (current != null)

{

if (current == speciesType) found = true;

current = (Species) unique.getNext();

}

if ( !found ) unique.addHead( speciesType );

speciesType = (Species) allSpecies.getNext();

}

allSpecies = unique;

}