**Question Set 1**

(a) Since 'Genus' is the superclass(parent class) of the 'Species' class, both the Genus & Species objects can use the variables & methods of the 'Genus' class

(b) Since 'Species' is the superclass(parent class) of the 'Specimen' class, both Species & Specimen objects can use the variables & methods of the 'Species' class

(c) Since 'Species' is the superclass(parent class) of the 'Specimen' class, both Species & Specimen objects can use the variables & methods of the 'Species' class

(d)

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

(e) This means that the programmer doesn’t have to re-write the codes of the parent class in the child class, as the child class inherits all the methods/variables of the parent class. Unless these inherited methods will be overridden.

(i) This is because since the ‘Genus’ is the superclass of ‘Species’ class, the ‘Species’ class will automatically inherit all the methods and variables of the ‘Genus’ class. Thus, it is possible for the ‘toString’ method to return the ‘getGenusName’ as it is from the parent class, and the ‘speciesName’ as its from the local class.

(ii) Inheritance

**Question Set 2**

(a) Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties’ direct access to them

(b)  functionality is defined in one place and not in multiple places, thus code is not repetitive and easily understood by other programmers, also, data inside our object is not modified unexpectedly by external code in a completely different part of our program, which ensures the confidentiality of the variables, values and methods.

(c) getCage()

(d) name

(e)

public class Genus

{

private String genusName;

Genus(String genusName)

{

this.genusName = genusName;

}

public void setGenusName(String genusName)

{

this.genusName = genusName;

}

public getGenusName()

{

return genusName;

}

public String toString()

{

return “Genus Name: “ +getGenusName();

}

}

(f) Advantage: The code will be more efficient and shorter as the ‘Specimen’ class will inherit all the members(variables & methods) of the ‘Species’ class. This means that the variables of ‘Species’ class do not have to be re-instantiated in the Specimen class.

Disadvantage: The disadvantage of inheritance is that since the child class inherits all the methods & variables of the superclass, when we want to modify the methods and variables of the superclass, the methods and variables of the child class will also be affected. Therefore these classes cannot be independent

**Question Set 3**

(a) Create a child class to inherit all the variables and methods of the ‘Specimen’ class. Then, instantiate some variables in this class that includes the description of each of the animal’s individual markings as required. Use methods so users can access/modify these variables (for different animals, different description). Finally, at the end, create a toString() method to return these values.

(b)

public void countSpeciments( Specimen[] animals, Species s)

{

System.out.print(“The number of specimens of “ + s + “ in the zoo is “ + animals.length)

}

(c)

listSpecies(Specimen[] animals)

{

//create an array of size 100 in case to make sure it’s enough

String species[] = 100;

int speciesIndex = 0;

boolean speciesAlreadyInArray = false;

for(i = 0 to animals.length-1)

{

for(j = 0 to 100)

{

If(species[j] == animals[i].getSpeciesName())

{

speciesAlreadyInArray = true;

}

}

If(speciesAlreadyInArray == false)

{

species[speciesIndex] = animals[i].getSpeciesName();

speciesIndex= speciesIndex+1;

speciesAlreadyInArray = false;

}

}

print(“All the different species are: “)

for(i = 0 to speciesIndex)

{

print(species[i])

}

}

**Question Set 4**

(a) ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.

(b)

LinkedList makeList( Specimen[] animals)

{

LinkedList linkedSpecimen = new LinkedList(Arrays.asList(animals));

}

(c)

LinkedList makeSpeciesList( LinkedList animals)

{

LinkedList allSpecies = new LinkedList(Arrays.asList(animals.getSpeciesName))

}

(d)

LinkedList makeSpeciesListUnique( LinkedList allSpecies)

{

//create an array of size 100 in case to make sure it’s enough

String speciesUnique[] = 100;

int speciesUniqueIndex = 0;

boolean speciesAlreadyInArray = false;

for(i = 0 to animals.length-1)

{

for(j = 0 to 100)

{

If(species[j] == animals[i].getSpeciesName())

{

speciesAlreadyInArray = true;

}

}

If(speciesAlreadyInArray == false)

{

speciesUnique[speciesUniqueIndex] = allSpecies[i];

speciesUniqueIndex= speciesUniqueIndex+1;

speciesAlreadyInArray = false;

}

}

LinkedList linkedSpecies = new LinkedList(Arrays.asList(speciesUnique))

}