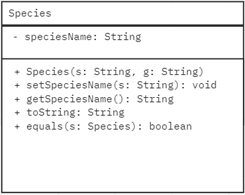
Question set 1

1. The genus class and species has a parent child(IS-A) relationship, where the genus class is the parent class to the species class.
2. The specimen and species class has an aggregate relationship (HAS-A) specifically a composition relationship, as a specimen needs a species within the constructor of its class.



1. -The team can reuse the methods within the genus class within the species class allowing the code to be less redundant.

-Readability from duplicate variables and methods would be easier.

1. -This does not cause an error as the species class overrides the same method from the parent class

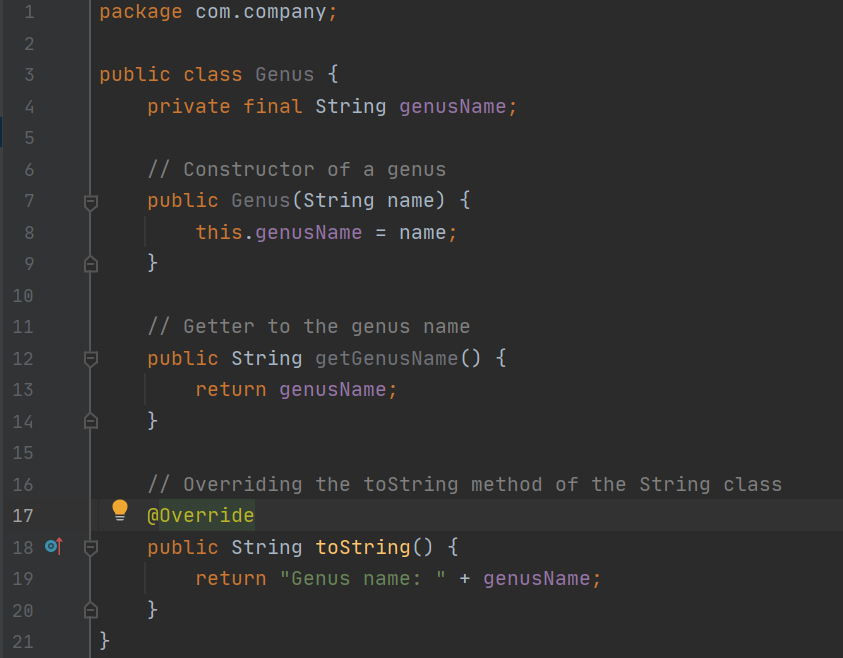
- Overriding

Question set 2

1. Encapsulation is wrapping data and its corresponding methods into one singular unit, in other words within a class variables are set to private, whilst setter and getter methods of the variables are made public.
2. -The fields of a class can only be read-only or write-only

-A class can have total control over what is stored in its fields.

1. public String getName()
2. private int cageNumber



1. **Advantage**: Methods and variables within the species class can be accessed within the specimen class

**Disadvantages**: A specimen cannot be created without a species

Question set 3

1. - Create a private string variable labeled marking

- Then add an additional string parameter within the constructor

-Make a setter and getter methods to change or get the marking

1. Public int countSpecimens( LinkedList<Specimen> animals, Species s )

{

Int sameSpecies = 0;

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

{

if(s.equals(animals.get(i).getTOA()))

{

sameSpecies +=1;

}

}

return sameSpecies;

}

1. listSpecies( Specimen[] animals )

{

String[] species

for(int i is less then the length of the size of the animals)

{

if (specimen’s species not in species[])

{

Add specimen’s species to species[]

}

else

{

pass

}

}

}

Question set 4

1. -It exports a set of operations.

-It exports a type.

1. LinkedList makeList( Specimen[] animals )

{

LinkedList <Specimen> animalsList = new LinkedList<Specimen>();

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

{

animalsList.add(animals[i]);

}

return animalsList;

}

1. LinkedList makeSpeciesList( LinkedList animals )

{

LinkedList <Specimen> speciesList = new LinkedList<Specimen>();

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

{

speciesList.add(animals.get(i).getTOA());

}

return speciesList;

}

1. LinkedList makeSpeciesListUnique( LinkedList allSpecies )

{

LinkedList <Specimen> uniqueSpeciesList = new LinkedList<Specimen>();

for(int i =0; i<allSpecies.size(); i++)

{

if(uniqueSpeciesList.isEmpty())

{

uniqueSpeciesList.add(allSpecies.get(i))

}

else

{

for(int j =0; j<uniqueSpeciesList.size(); j++)

{

if(uniqueSpeciesList.get(j).equals(allSpecies.get(i))

{

;

}

else

{

uniqueSpeciesList.add(allSpecies.get(i))

}

}

}

}

return speciesList;

}