Option D — Object-oriented programming

10. (a) The `Species` class inherits from the `Genus` class;
Accept "`Species` extends `Genus`". ***[1 mark]***

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

(c) *Award up to **[4 marks max]**.*
Correct visible 3 tiered diagram *eg* box diagram;
Correct class name in top box; *Accept variations eg. "Species extends Genus".*
Correct member variable in middle box;
ALL Correct member functions in lower box;

```
Species
String: speciesName
setSpeciesName(String s)
String getSpeciesName()
String toString()
boolean equals(Species s)
```

[4 marks]

(d) *Award **[1 mark]** for identifying a benefit and **[1 mark]** for an elaboration, for **two** benefits, up to **[4 marks max]***
Benefits:
Avoids duplicate code;
Simplifies testing;
Faster development;
Can mimic real-world object relationships;
Accept other reasonable benefits

Example:
Avoids duplicate code;
Since all functionality of the parent class is inherited, it is not necessary to duplicate the code to reproduce that functionality in the new class;
Simplifies testing;
Functionality inherited from a parent class does not need to be re-tested in the new class; ***[4 marks]***

(e) (i) The two methods are related to the specific class declared;
The complier selects the correct method; ***[2 marks]***

(ii) Polymorphism **OR** overriding;
*Award **[1 mark]** for either.* ***[1 mark]***

11. (a) Encapsulation refers to the practice of hiding the structure and representation of data within a class and making it accessible outside that class via accessor functions;

Accept "Refers to bundling the data and methods that operate on the data into one unit the methods and data can be accessed via dot notation or accessor methods." ***[1 mark]***

(b) *Award **[1 mark]** for identifying a benefit and **[1 mark]** for an elaboration, for two benefits, up to **[4 marks max]**.*
Advantages:
Data hiding and security [AM]
Ease of maintenance;
Ease of testing;
Speed of development;

Accept other reasonable advantages.

Example:
Advantage: Ease of testing;
By putting all the structure of the data in a single class, other classes which make use of that data can be easily tested by simply providing them with data values even before the "real" data class is available for testing;

Advantage: Ease of maintenance;
If there are changes to the data, or how it is stored or accessed internally, only the one class needs to be changed to accommodate that. All the other classes will continue to use the public accessor methods and will not need to be changed; ***[4 marks]***

(c) Any accessor method in the `Specimen` class;
Example: `getName()`, `getCage()`, `getTOA()` are accessor methods in the `Specimen` class. ***[1 mark]***

(d) Any instance variable in the `Specimen` class;
Example: `cageNumber`, `name` and `TOA` are instance variables in the `Specimen` class. ***[1 mark]***

(e) *Award up to **[3 marks max]**.*
*Award **[1 mark]** for a constructor method requiring a single string as a parameter;*
*Award **[1 mark]** for* `String getGenusName()` *and a set method for the genus name;*
*Award **[1 mark]** for* `String toString()` *method that returns a valid string;*

Example:

```
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;
     }
}
```

[3 marks]

(f) *Accept any reasonable advantage and disadvantage. Award **[1 mark]** for identifying the (dis)advantage and **[1 mark]** for an elaboration, for one advantage and one disadvantage, up to **[4 marks max]**.*

Example:

Advantage: The `Specimen` objects would inherit all the attributes of the `Species` object;
This would allow code in the `Specimen` object to access species-related methods directly rather than having to invoke them on a `Species` object using dot-notation;
Disadvantage: Logically inconsistent, in that data in the Species class may not be consistent across the associated Specimens;
Potential data duplication / waste of memory space;

Example 2:

Advantage: The Specimen object, as a subclass, will inherit all methods and instance variables applying to the Species object;
This would take up less space and make access to all common methods between specimens easier;
Disadvantage: Not all methods and variables are consistent between all specimens;
This means that changes made to methods or variables in the Species class may not apply to all specimens; ***[4 marks]***

12. (a) The markings are a characteristic of an individual animal;
Therefore the description of markings should be an instance variable(s) within the `Specimen` class;
There should be accessor (get/set) methods for the markings;
The `toString()`method should include the description of the markings;
*If candidate creates a new class for the markings then award marks appropriately, **[2 marks max]**, for the variables.* ***[4 marks]***

.

(b) *Award up to **[8 marks max]**.*
*Award **[1 mark]** for correct method declaration (accept void or int as method type);*
*Award **[1 mark]** for initializing species count to zero;*
*Award **[1 mark]** for correctly determining array length;*
*Award **[1 mark]** for correctly looping through the array elements;*
*Award **[1 mark]** for retrieving species object from each specimen;*
*Award **[1 mark]** for comparing species object retrieved with species being counted using equals();*

> ***Note:** To earn this mark, the student must either use the equals method of the Species class or use the getSpeciesName() method to retrieve the name as a string and then use the String class equals() method.*

*Award **[1 mark]** for incrementing species count correctly;*
*Award **[1 mark]** for outputting species count (accept "output" in lieu of System.out.println);*

Example:

```
void countSpecimens( Specimen[] animals, Species s )
{
     int sCount = 0;
     int i;
     for (i=0; i< animals.length; i++)
     {
        if (s.equals(animals[i].getTOA())
        OR
        if(s.getSpeciesName().equals(animals[i].getTOA().getSpeciesName()))

        sCount++;
     }
     System.out.println( sCount );
}
```

[8 marks]

(c) *Award up to **[6 marks max]**.*
*Award **[1 mark]** for initialisation of variables;*
*Award **[1 mark]** for appropriate use of loops and selection*
*Award **[1 mark]** for creating something to hold a list of unique species;*
*Award **[1 mark]** for looping through the input array of Specimen objects;*
*Award **[1 mark]** for searching the list of unique species for each animal's species;*
*Award **[1 mark]** for inserting the animal's species into the list if it is not there already;*
*Award **[1 mark]** for producing the list of unique species as output;*

***Note:** In this question the term "pseudocode" indicates that syntax is not being tested; the emphasis is on the algorithm.*

Example 1:

```
//ANIMALS is an array containing Specimen objects
//UNIQUE is a collection, initially empty
//SPECIES is a Species object

loop BEAST from 0 to the number of elements in ANIMALS
     NEW_SPECIES = true
     UNIQUE.resetNext()
     loop while UNIQUE.hasNext()
          if UNIQUE.getNext() = species of ANIMALS[BEAST] then
               NEW_SPECIES = false
          end if
     end loop
     if NEW_SPECIES then
          UNIQUE.addItem( species of ANIMALS[BEAST] )
     end if
end loop

UNIQUE.resetNext()
loop while UNIQUE.hasNext()
     SPECIES = UNIQUE.getNext()
     output SPECIES.toString()
end loop
```

***Note:** The output does not need to be generated in a second pass after finding all the unique species. An alternative would be to generate the output for each new unique species as it is found.*

An alternative approach would be to pass through the array, and test each object. Using a second array into which the unique species name is stored as these are found. ***[6 marks]***