COM102 (Object Oriented Programming)

Coursework - Practical Skills Assessment 2

Title: Pet Clinics Management System

Due: Noon, 12:00 on Thursday, 17 April 2025 (Week 12)

This coursework constitutes 60% of the total marks for the module. Feedback will be made available on Blackboard within 20 working days from the submission deadline.

For this assignment， you are required to work individually.

By making a submission, you will be deemed to have made the following declaration of ownership. Source: https://www.ulster.ac.uk/student/exams/policies-procedures.

"I declare that this is all my own work. Any material I have referred to has been accurately referenced and any contribution of Artificial Intelligence technology has been fully acknowledged. I have read the University's policy on academic misconduct and understand the different forms of academic misconduct. If it is shown that material has been falsified, plagiarised, or I have otherwise attempted to obtain an unfair advantage for myself or others, I understand that I may face sanctions in accordance with the policies and procedures of the University. A mark of zero may be awarded and the reason for that mark will be recorded on my file."

Overview

This assignment is concerned with the use of collection to store and manage objects. Write a Java program to store and manage data from a set of pets registered at a pet clinic according to the following requirements:

- **Functional Requirements**

  1. The application must be able to handle data from at least two types of pets such as cats and dogs.
  2. For each pet, their name, age, colour, weight, and breed should be recorded.

     Hint: consider creating an array of type `Pet`, where `Pet` is a base class you create containing the name, age, colour, and weight as instance variables. Each sub class represents each type of pet has an additional instance variable breed that can store the particular breed of a pet. For example, it could be Persian, Tabby, etc. for a cat and Spaniel, terrier, etc.. for a dog. You need to check that each variable has been entered correctly in whatever format you specify. You might also want to inform the user that if any of the data fields for a given pet are left blank, that pet's full details will NOT be saved beyond the current session.

3. The program should contain a method called `speak( )` that returns a typical animal noise, plus a description of the animal such as ...

   "Miaow! I am Pixel, a 4 year old tabby", or

   "Woof! I am Rex, a 9 year old terrier".

4. The program needs to enable pets to be **added** to the clinics or **deleted** from the clinics. It should also have any facility to **modify** pet details after they have been created.

   Hint: deleting data from an array may leave a blank space in the array, so keep a count of the total number of pets and search for the next free array cell when adding another pet.

5. It must be possible to **report** (i.e. print to the screen) on the clinic. This report should indicate the clinics name which could be hard-coded, the total number of each type of pets registered and the dominant color for these pets.
6. The program should allow the user to **view** all the pets currently registered at the clinic.
7. At the end of a session, when the program is being terminated, the clinics details along with the details for each registered pet should be **written** to disk. These details should be recorded in two standard text files: one containing clinics details (called "ClinicsDetails.txt") and the other holding all the pet records, to be called "PetDetails.txt".

   Hint: don't save blank records to disk, check that there is a valid "pet" object in each array cell before writing its contents to disk. If any of the fields within a pet's details have been left blank (string length of zero) then that student is invalid and their details should **not** be written to disk.

8. When the program is started, it should read from these standard text files to **re-populate** the application with any previously stored data as a starting point.

   Hint: each record requires a new instance of the Pet class to be created and linked to the next available free cell in your array.

9. The program user must also be able to **search** for a pet by name or color, causing the sought pet's details to be displayed and its `speak( )` method to be called if present.

10. Any additional features not mentioned above – to exercise your creativity, you are welcomed to implement any additional features/enhancement

11. The program must employ a **console** interface only.

You must develop a set of test cases and at least one unit test and record the results of applying these tests to your software.

- **Other assignment requirements:**

  1. The adoption of object-oriented principles should be evident in your implementation of the above requirements including the use of inheritance and polymorphism.
  2. A structured approach to testing should be evidenced by submission of a test plan including at least **one** unit test and outcomes in accordance with the above requirements.

Submission

- The following 3 deliverables must be submitted to **Blackboard** on or before **noon on Thursday, 17 April 2025** (Week 12)

  1. **Source code** – A Zip file containing all code developed for the application. The easiest way to create this is to Zip the contents of the **src** folder of your Java project. Each Java file should also be provided as a PDF document.

  2. **Design and Development Document** – A PDF document (no more than **600** word) that contains a written account of the design and development decisions made during the development of your application. It must contain the following sections:

     - Evidence of the adoption of object-oriented principles
       - Use of UML diagrams
       - Inheritance hierarchy design
       - Demonstration of polymorphism
     - Consideration of data structures and program control structures used with justification for your decisions.

  3. **Testing Document** – A PDF document (no more than **600** words) that describes the approach to testing and implementation and test cases used. It must include the following.

     a. A table summarizing all the test cases designed as illustrated in Table 1 along with a set of screenshots to support each test case, which should include:

        - Test case ID
        - Test case description, outlining what it is designed to do
        - Test data defining any necessary data required – Please include a **screenshot** showing the test data input for the given test case.
        - Expected result
        - Actual result – Please include a **screenshot** along with some description to show the actual results of the test
        - Status (Pass/Fail) – Wherever appropriate, include **screenshots and explanation** to verify whether the test has passed/failed.

     It is expected that for each requirement, it should also test 2 ~ 3 boundary/error conditions in addition to normal condition.

Table 1 A summary of test cases designed

Test case NO	**Test case description**	**Test data**	**Expected results**	**Actual results**	**Status (Pass/Fail)**
TC-01	Test for adding a pet	xxx as shown in Fig. x.x		xxx as shown in Fig. x.x	Fail as illustrated in Fig. x.x
TC-02					
….					

b. The results of your unit tests implemented including

- The purpose of the testing method you implemented. What behaviour or functionality did you test? What should be the expected output?
- Screen capture of code snippet of the test method(s) implemented
- The results of your unit tests.

c. A summary table (Table 2) outlining the implementation of each requirement and test case IDs designed for testing each requirement must be also included in your report otherwise **marks will be deducted from your overall score (- 5%)**.

Table 2 A summary of the implementation of each requirement

Requirement	**Implemented (Yes/No)**	**Test case ID designed**
Ability to pre-populate the application with any previously store data		
Ability to add a pet		
Ability to delete a pet		
……		
Implementation of any additional features		

4. A copy of your **vodcast** must be submitted on or before **noon, 12:00 on Thursday 17th April 2025** (Week 12). The vodcast which provides a brief walk-through of your code should demonstrate clearly the features you have created as well as a clear readable review of the code that make the features work. It should be encoded in mp4 format and last no more than 5 minutes. Please note that the information contained in your vodcast will be used to support the assessment of your implementation of each feature. **You will be marked down if no relevant information is provided as required.**

5. According to Ulster University Assessment Code of Practice, where submitted work exceeds the agreed assessment limit, a margin of up to **+10%** of the work limit will be allowed without any penalty of mark deduction. **If the work submitted is significantly in excess of the specified limit (+10%), there is no expectation that staff will assess the piece beyond the limit or provide feedback on work beyond this point**. Markers will indicate the point at which the limit is reached and where they have stopped marking. A mark will be awarded only for the content submitted up to this point. No additional deduction or penalty will be applied to the overall mark awarded. The student is self-penalising as work will not be considered/marked.

Assessment Criteria:

- Application of concepts - Structured, commented and readable code and appropriate use of variables. Implementation of suitable program control structures, classes and methods to solve problem (25%)
- Program executes according to specification (25%)
- Structured approach to testing (20%)
- Design and planning document (20%)
- Vodcast (10%)

Marking rubric

Criteria **(100%)**	**0-39%** **fail**	**40-49%** **3rd**	**50-59%** **2.2**	**60-69%** **2.1**	**70-100%** **1st**
Application concepts (25%)	Poor use of variables and lacking comments. Poor layout. Submitted code does not display object-oriented structure. No use of a collection manager class and no data validation attempt	Variable names are largely appropriate. Sparsely commented. Limited and poorly structured code. Submitted code does not display enough object-oriented structure. Limited use of a collection manager class and limited data validation.	Satisfactory use of Variables and comments Limit but structured code. Submitted code displays satisfactory object-oriented structure. Satisfactory use of a collection manager class and data validation.	Good use of variables and comments. Good layout and structured code. Submitted code displays good object-oriented structure. Good use of a collection manager class and data validation.	Excellent use of variables, comments and white spaces. Clear, consistent layout and well-structured code. The adoption of object-oriented principles is very clearly evident in the implementation. Excellent use of a collection manager class. Data validation well implemented
Program execution according to specification (25%)	Unable to demonstrate working program. Submitted code will not execute	Submitted code runs and performs a basic task from the specification.	Submitted code runs and performs basic tasks from the specification.	Submitted code runs and perform most tasks from the specification.	Submitted code runs with no modification required and fully meets the specification.
Structured approach to testing (20%)	Very limited solution with little and no testing results.	Test cases only covering 2-3 requirements. Test outcome poorly explained. No unit tests	Test cases covering most requirements. Test outcome satisfactorily explained. Unit tests provided but test methods could be better written	Test cases covering all requirements and some boundary/error conditions. Test outcome well explained. Unit tests well performed	Test cases very well designed and very clearly described, covering all requirements and boundary and error conditions. Test outcome very well explained. Excellent use of unit tests
Design and development document (20%)	No evidence of the adoption of OOP. Poor justification for data structure used	Limited evidence of the adoption of OOP and limited justification of data structure and control structure used	Satisfactory evidence of the adoption of OOP and reasonably good justification of data structure and control structure used	Good evidence of the adoption of OOP and good justification of data structure and control structure used	Excellent evidence of the adoption of OOP and the well consideration of data structure and control structure with comprehensive justification
Vodcast (10%)	Poor adherence to vodcast guidelines. Lacking clarity & not able to discuss main issues	Limited adherence to vodcast guidelines. Limited discussion of main issues.	Satisfactory adherence to guidelines but perhaps rushed or verbose presentation. Able to broadly discuss what was done. High-level discussion of the program limitations	Close adherence to the guidelines, well paced and clearly presented. Able to discuss the main code segments and aware of the main limitations.	Close adherence to the guidelines, well-paced and clearly presented. Able to clearly articulate depth of knowledge. *Able to discuss the main limitations and discuss how challenges were overcome.*

Vodcast Guidelines

Note: The total duration of the recording should not exceed 5 minutes

The vodcast must include a screencast of the program demo and code walkthrough, and a live view of the presenter, positioned in the bottom right hand corner of the recording[1]

Please note that your face must be shown on the screen during the video – If you fail to do so, you may be invited to an interview after the submission.

Program Demo Walkthrough

Evidence of completeness, confidence in operation, testing and awareness of any limitations

1. Execute your program within the chosen IDE and present an overview of the main functions available.
2. Demonstrate the implementation of the following features one by one, discussing the program output to confirm the correct operation.
   - When the program is started, it reads from three csv files to populate the application with any previously stored data as a starting point.
   - A menu appears after the program is started, allowing the user to perform the following operation
     a. **Add** a pet to the clinic
     b. **Modify** the detail of a pet in the clinic an existing loan -
     c. **Delete** a pet from the clinic

        For the above operations, i.e. Add/Delete/Modify, you need to demonstrate your program with each information correctly entered in whatever format you specify AND with any of the data fields entered in a wrong format or left blank

     d. **View** all the pets currently registered at the clinic
     e. **Report** on the clinic
     f. **Search for** a pet by name or color

        Demonstrate your program search for a pet registered at the clinic

        Demonstrate your program search for a pet that has not yet been registered.

     g. **Demonstrate the implementation of any additional features**

     When the program exists, the clinics details along with the details for each registered pet should be written to disk as required.

2. Demonstrate known issues / bugs that exist within the program and indicate whether these were due to technical complexity, lack of time or some other reason.
3. Provide self-assessment of how well you feel you have achieved the main elements of the assignment

[1] Several free tools support this dual recording feature e.g. https://screencast-o-matic.com/

Source Code Walkthrough

Application of concepts and evidence of understanding of code structure and key OOP concepts

1. Present an overview of the application you have implemented
2. Discuss the approach adopted for the implementation of each task.
3. Ensure the code is well organised and structured.
4. Discuss the code/concepts behind each function/feature presented within the vodcast
5. Evidence your understanding of key OOP concept such as the use of inheritance/polymorphism etc.