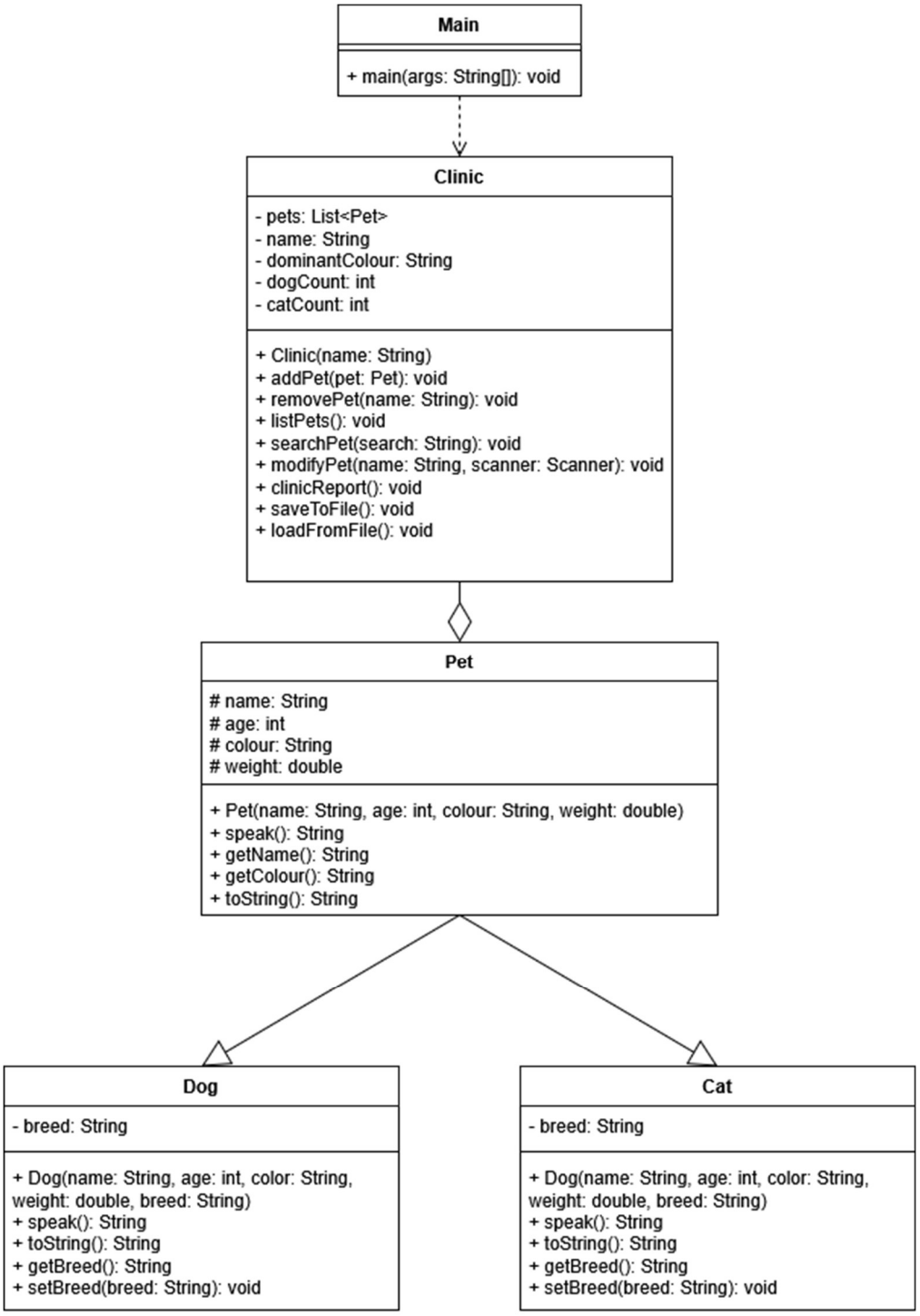
COM102 Object Oriented Programming Coursework 2

James Law – B00983387

UML Diagram



The Pet class defines the fundamental attributes common to all pets: name (String), age (int), colour (String), and weight (double). It also specifies common behaviors through methods such as Pet() (constructor), speak() (String), getName() (String), getColor() (String), and toString() (String).

The Cat class inherits from Pet and extends it with the breed (String) attribute, specific to cats. It overrides the speak() and toString() methods to provide cat-specific behavior. Additional methods include Cat() (constructor), getBreed() (String), and setBreed() (String).

Similar to the Cat class, the Dog class inherits from Pet and adds the breed (String) attribute, specific to dogs. It also overrides the speak() and toString() methods for dog-specific behavior. It includes methods Dog() (constructor), getBreed() (String), and setBreed() (String).

The Clinic class serves as the central management component, responsible for maintaining a collection of Pet objects using an ArrayList. It provides functionalities for managing pets through methods such as Clinic() (constructor), addPet() (Pet), removePet() (String), listPets(), searchPet() (String), modifyPet() (String, Scanner), clinicReport(), saveToFile() (throws IOException), and loadFromFile().

The Main class contains the main() method, serving as the application's entry point, and provides the user interface to access the system.

Inheritance Hierarchy Design

The Pet class is the base class, also known as the superclass. It defines the attributes and behaviours common to all pets, providing a foundation for more specialized pet types.

The Cat and Dog classes are derived from the Pet class and are therefore considered subclasses. They inherit the attributes and methods of the Pet class, extending them with their specific characteristics. For example, the Cat and Dog classes each include a breed attribute and provide their implementations of the speak() method.

Demonstration of Polymorphism

Polymorphism is a key feature of object-oriented programming, and it is effectively demonstrated in this system through the ‘speak()’ method. The Pet class declares the ‘speak()’ method, establishing a common interface for all pet types to provide a verbalization. The Cat and Dog classes then override this ‘speak()’ method, providing specialized implementations appropriate to the specific animal. A Cat object's ‘speak()’ method returns "Meow!...", while a Dog object's ‘speak()’ method returns "Woof!...". This

polymorphic behaviour allows the Clinic class to interact with Pet objects without being concerned with their specific type.

Data Structures and Program Control Structures

The system employs specific data and program control structures to optimise data management and program flow.

The Clinic class utilises an ArrayList named ‘pets’ to store a dynamic collection of Pet objects. The ArrayList is selected for its flexibility in handling various pets. It allows for efficient addition and removal of pets, and its dynamic sizing eliminates the limitations associated with fixed-size arrays.

Within the ‘clinicReport()’ method, a HashMap named ‘colourCounts’ is employed to track the frequency of different pet colours efficiently. The HashMap provides key-value pairs, where the colour string is the key, and the integer count is the value. This structure enables efficient lookups and updates of colour counts.

Program Control Structures:

Switch statements are used within the ‘modifyPet()’ method to manage the different choices available to the user when modifying pet attributes. This provides a structured and readable way to handle multiple modification options.

‘For loops’ are used to iterate over the collection of Pet objects stored in the ArrayList, to allow for operations such as listing all pets, searching for a specific pet, and generating the clinic report.

‘While loops’ manage the main program loop, providing a continuous interactive session until the user exits.

These data and control structures are selected to combine efficient data storage, clear and predictable program flow, and effective user interaction.