In Partial Fulfillment of the Requirements for the

CS 223 - Object Oriented Programming

**Pet Information System Documentation**

Presented to:

**Dr. Unife O. Cagas**

Professor

Presented by:

**Teriote, Nice Kyorie**

BSCS-2A: Computer Science

May, 2024

**Pet Information System Documentation**

**Description**

The Pet Information System is a Python-based application designed to store, manage, and display information about various types of pets. The system showcases essential object-oriented programming (OOP) principles such as inheritance, polymorphism, abstraction, and encapsulation. This project is intended for both educational purposes and practical use, providing a user-friendly interface for pet owners to record and retrieve information about their pets.

**Objective**

**Demonstration of Object-Oriented Programming (OOP) Concepts:**

•To illustrate key OOP principles such as inheritance, polymorphism, abstraction, and encapsulation through practical implementation in a Python application.

**User-Friendly Interface:**

•To offer an intuitive and easy-to-use text-based interface that allows users to input and retrieve pet information effortlessly.

**Extendibility and Flexibility:**

•To design a flexible system architecture that can be easily extended to include additional types of pets or more detailed pet information in the future.

**Accurate and Detailed Record Keeping:**

•To enable accurate and detailed record-keeping of pet attributes such as name, age, breed, color, and water type, ensuring all relevant information is captured.

**Consistent and Standardized Output:**

•To provide consistent and standardized output regarding pet details and behaviors, ensuring users receive clear and uniform information.

**Importance and contribution**

**Application:**

•Beyond its educational value, the system has practical applications for pet owners, veterinarians, and pet care facilities. It provides a convenient way to manage pet records, making it a useful tool for real-world scenarios.

**Code Reusability and Maintenance:**

•By utilizing inheritance, the system promotes code reusability and simplifies maintenance. Common functionalities are defined in the base class Pet, while specific details are handled by subclasses (Dog, Cat, Fish). This modular approach reduces code duplication and enhances maintainability.

**Encouraging Innovation:**

•The extendable nature of the system encourages developers to innovate and expand the application. By providing a solid foundation, it inspires further development and creative enhancements, fostering a culture of innovation.

**Four principles of Object Oriented Programming**

**1. Encapsulation:** demonstrated through the use of private attributes and public getter methods.

class Pet(ABC):

def \_\_init\_\_(self, name, species, age):

self.\_\_name = name

self.\_\_species = species

self.\_\_age = age

@property

def name(self):

return self.\_\_name

@property

def species(self):

return self.\_\_species

@property

def age(self):

return self.\_\_age

**2. Abstraction:** achieved by defining an abstract base class and abstract method.

class Pet(ABC):

@abstractmethod

def speak(self):

pass

**3. Inheritance:** shown by creating subclasses (Dog, Cat, Fish) that inherit from a common superclass (Pet).

class Dog(Pet):

def \_\_init\_\_(self, name, age, breed):

super().\_\_init\_\_(name, "dog", age)

self.\_\_breed = breed

def speak(self):

return "Woof!"

@property

def breed(self):

return self.\_\_breed

class Cat(Pet):

def \_\_init\_\_(self, name, age, color):

super().\_\_init\_\_(name, "cat", age)

self.\_\_color = color

def speak(self):

return "Meow!"

@property

def color(self):

return self.\_\_color

class Fish(Pet):

def \_\_init\_\_(self, name, age, water\_type):

super().\_\_init\_\_(name, "fish", age)

self.\_\_water\_type = water\_type

def speak(self):

return "Glub glub!"

@property

def water\_type(self):

return self.\_\_water\_type

**4. Polymorphism:** illustrated through the speak method, which is implemented differently in each subclass but can be called uniformly on any Pet object.

def main():

while True:

print("\nWelcome to the Pet Information System!")

choice = input("Enter 'dog', 'cat', 'fish', or 'quit' to exit: ").lower()

if choice == 'quit':

break

if choice == 'dog':

name, age, breed = get\_dog\_info()

dog = Dog(name, age, breed)

print(f"{dog.name} is a {dog.age}-year-old {dog.breed} {dog.species}.")

print(dog.name, "says:", dog.speak())

elif choice == 'cat':

name, age, color = get\_cat\_info()

cat = Cat(name, age, color)

print(f"{cat.name} is a {cat.age}-year-old {cat.color} {cat.species}.")

print(cat.name, "says:", cat.speak())

elif choice == 'fish':

name, age, water\_type = get\_fish\_info()

fish = Fish(name, age, water\_type)

print(f"{fish.name} is a {fish.age}-year-old {fish.water\_type} {fish.species}.")

print(fish.name, "says:", fish.speak())

else:

print("Invalid choice. Please enter 'dog', 'cat', 'fish', or 'quit'.")

**Hardware and software uses**

**Hardware uses**

**1. Computer or Laptop**

**2. Input Devices:**

**Keyboard:** For entering pet information and navigating the system.

**Mouse:** For interacting with the graphical user interface (if applicable).

**3. Output Devices:**

**Monitor:** For displaying the system interface and output.

**Printer (optional):** For printing pet information records.

**Software Requirements**

**1. Operating System**

**2. Programming Environment:**

**•Python Interpreter:** Version 3.6 or higher is required to run the Python code.

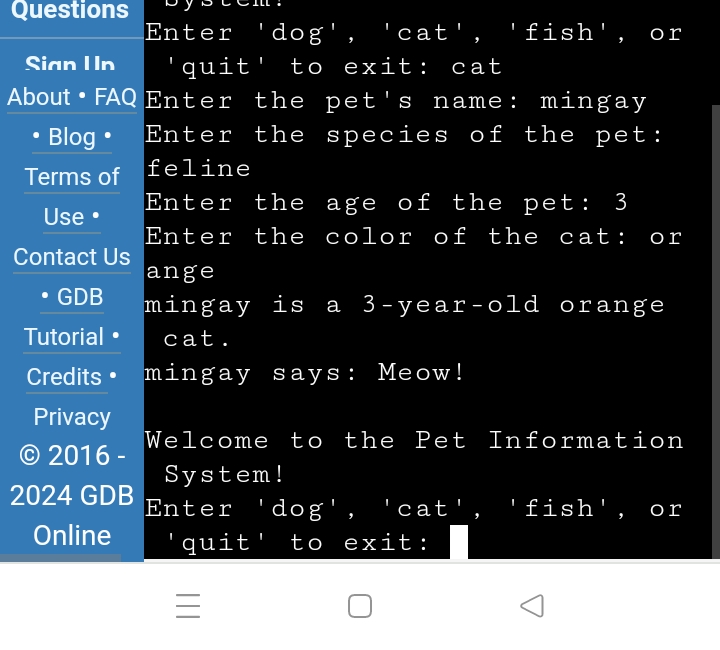
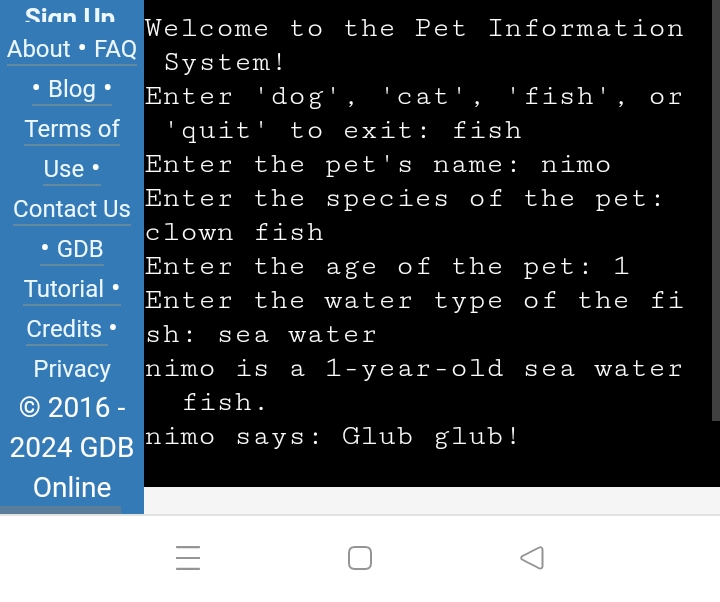
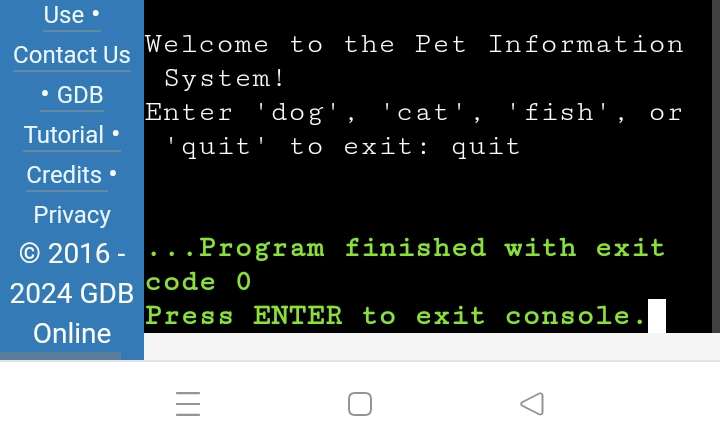
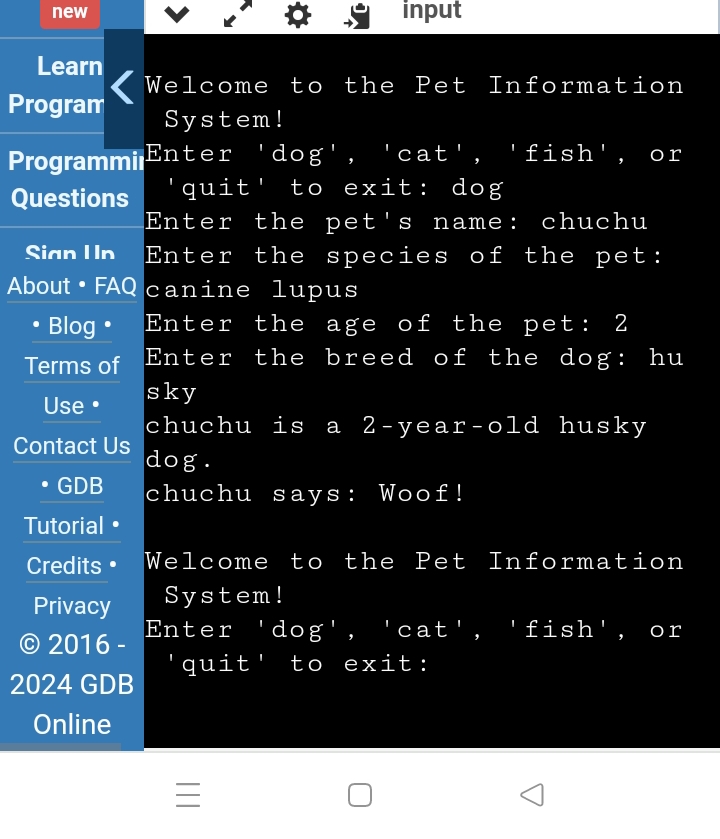
**•IDE/Text Editor:** Software such as PyCharm, Visual Studio Code, Sublime Text, or even the default Python IDLE can be used for writing and editing the code.

**3. Python Libraries:**

**•abc:** This is part of Python's standard library and is used for creating abstract base classes.

**•sys:** If handling system-specific parameters and functions, though not explicitly mentioned in the provided code.

**Output**



**Discretion**

The provided code for the Pet Information System effectively demonstrates the four fundamental principles of Object-Oriented Programming (OOP): encapsulation, abstraction, inheritance, and polymorphism. By running the program, users can experience how these principles are applied in practice.

**Code**

from abc import ABC, abstractmethod

# Encapsulation: Defining the Pet class with private attributes

class Pet(ABC):

def \_\_init\_\_(self, name, species, age):

self.\_\_name = name

self.\_\_species = species

self.\_\_age = age

# Abstraction: Defining an abstract method that must be implemented by subclasses

@abstractmethod

def speak(self):

pass

# Encapsulation: Getter methods to access private attributes

@property

def name(self):

return self.\_\_name

@property

def species(self):

return self.\_\_species

@property

def age(self):

return self.\_\_age

# Inheritance and Polymorphism: Dog class inherits from Pet

class Dog(Pet):

def \_\_init\_\_(self, name, age, breed):

super().\_\_init\_\_(name, "dog", age)

self.\_\_breed = breed

def speak(self):

return "Woof!"

@property

def breed(self):

return self.\_\_breed

# Inheritance and Polymorphism: Cat class inherits from Pet

class Cat(Pet):

def \_\_init\_\_(self, name, age, color):

super().\_\_init\_\_(name, "cat", age)

self.\_\_color = color

def speak(self):

return "Meow!"

@property

def color(self):

return self.\_\_color

# Inheritance and Polymorphism: Fish class inherits from Pet

class Fish(Pet):

def \_\_init\_\_(self, name, age, water\_type):

super().\_\_init\_\_(name, "fish", age)

self.\_\_water\_type = water\_type

def speak(self):

return "Glub glub!"

@property

def water\_type(self):

return self.\_\_water\_type

def get\_pet\_info():

name = input("Enter the pet's name: ")

species = input("Enter the species of the pet: ")

age = int(input("Enter the age of the pet: "))

return name, species, age

def get\_dog\_info():

name, \_, age = get\_pet\_info()

breed = input("Enter the breed of the dog: ")

return name, age, breed

def get\_cat\_info():

name, \_, age = get\_pet\_info()

color = input("Enter the color of the cat: ")

return name, age, color

def get\_fish\_info():

name, \_, age = get\_pet\_info()

water\_type = input("Enter the water type of the fish: ")

return name, age, water\_type

def main():

while True:

print("\nWelcome to the Pet Information System!")

choice = input("Enter 'dog', 'cat', 'fish', or 'quit' to exit: ").lower()

if choice == 'quit':

break

if choice == 'dog':

name, age, breed = get\_dog\_info()

dog = Dog(name, age, breed)

print(f"{dog.name} is a {dog.age}-year-old {dog.breed} {dog.species}.")

print(dog.name, "says:", dog.speak())

elif choice == 'cat':

name, age, color = get\_cat\_info()

cat = Cat(name, age, color)

print(f"{cat.name} is a {cat.age}-year-old {cat.color} {cat.species}.")

print(cat.name, "says:", cat.speak())

elif choice == 'fish':

name, age, water\_type = get\_fish\_info()

fish = Fish(name, age, water\_type)

print(f"{fish.name} is a {fish.age}-year-old {fish.water\_type} {fish.species}.")

print(fish.name, "says:", fish.speak())

else:

print("Invalid choice. Please enter 'dog', 'cat', 'fish', or 'quit'.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

#Encapsulation - Bex Tuychiev

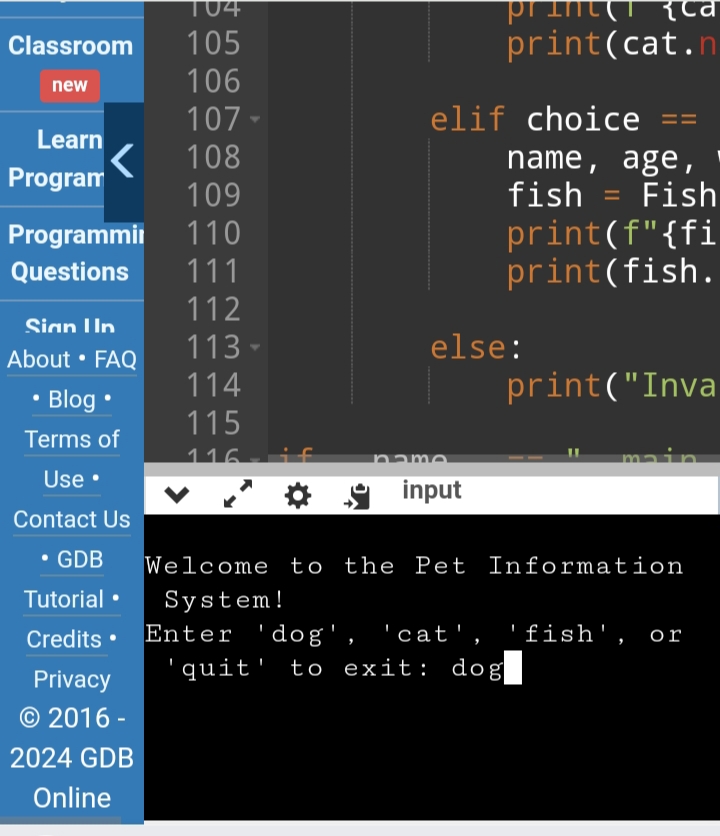
#Abstract - Nishita Tiwari

#Inheritance - Adam Spannbauer

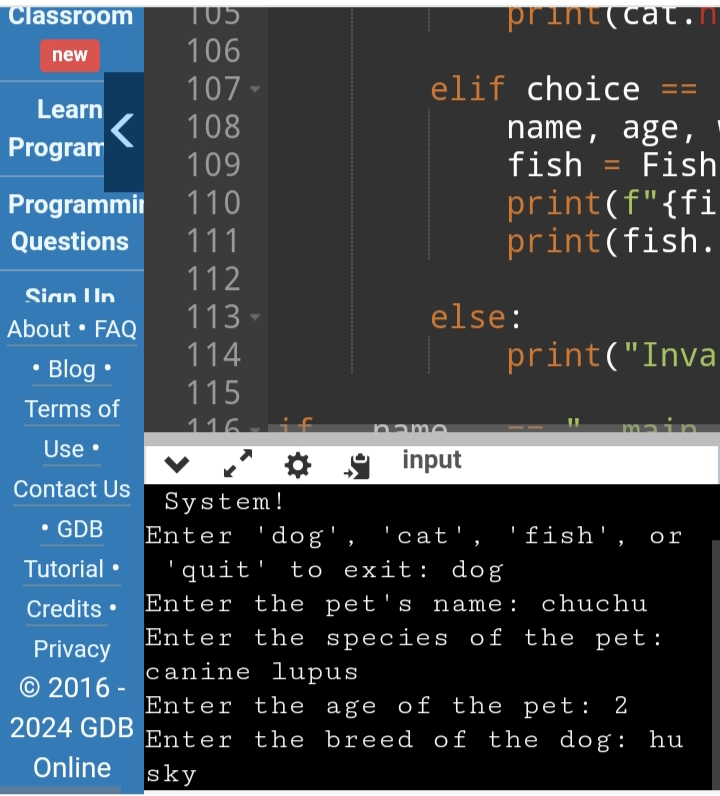
#Polymorphism - Nikita Duggal

**User Guide**

**Step - 1:** Choose the type of pet you have in your care.



**Step - 2:** Fill the the name, species, age, and breed of the dog you have.



**Reference**

Online GDB: <https://onlinegdb.com/3FQzAVA62>

Encapsulation: <https://www.datacamp.com/tutorial/encapsulation-in-python-object-oriented-programming>

Abstraction: [https://www.analyticsvidhya.com/blog/2024/02/understanding-abstraction-in-python-simplifying-complex-concepts/#:~:text=Abstraction%20in%20Python%20refers%20to,be%20shared%20among%20multiple%20classes.](https://www.analyticsvidhya.com/blog/2024/02/understanding-abstraction-in-python-simplifying-complex-concepts/" \l ":~:text=Abstraction%20in%20Python%20refers%20to,be%20shared%20among%20multiple%20classes.)

Inheritance: [https://www.datacamp.com/tutorial/super-multiple-inheritance-diamond-problem?utm\_source=google&utm\_medium=paid\_search&utm\_campaignid=19589720824&utm\_adgroupid=157156376311&utm\_device=m&utm\_keyword=&utm\_matchtype=&utm\_network=g&utm\_adpostion=&utm\_creative=698229374827&utm\_targetid=dsa-2218886984100&utm\_loc\_interest\_ms=&utm\_loc\_physical\_ms=9061352&utm\_content=&utm\_campaign=230119\_1-sea~dsa~tofu\_2-b2c\_3-row-p2\_4-prc\_5-na\_6-na\_7-le\_8-pdsh-go\_9-na\_10-na\_11-na-may24&gad\_source=1&gclid=CjwKCAjwupGyBhBBEiwA0UcqaJNB40zD0QNTj66zAaIi-F0xiWpmmGenbPCCd2xZiHT\_jQmrVpWusBoCWbYQAvD\_BwE](https://www.datacamp.com/tutorial/super-multiple-inheritance-diamond-problem?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720824&utm_adgroupid=157156376311&utm_device=m&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=698229374827&utm_targetid=dsa-2218886984100&utm_loc_interest_ms=&utm_loc_physical_ms=9061352&utm_content=&utm_campaign=230119_1-sea~dsa~tofu_2-b2c_3-row-p2_4-prc_5-na_6-na_7-le_8-pdsh-go_9-na_10-na_11-na-may24&gad_source=1&gclid=CjwKCAjwupGyBhBBEiwA0UcqaJNB40zD0QNTj66zAaIi-F0xiWpmmGenbPCCd2xZiHT_jQmrVpWusBoCWbYQAvD_BwE&fbclid=IwZXh0bgNhZW0CMTAAAR09jRrPpPSMPmqsA2ySa5Gj71pK2-U8DcLyYgRTNQ8DZTY1Tjbz_8-SVZw_aem_AQegIatolpUdTRi8IG0LPEVi4SMD6kWridb4nL6W5pzFjhRoTgJBsdTre8zf1n8VMcikyNrSDDJPVXxLhiTSsHYZ" \t "https://www.facebook.com/_blank)

Polymorphism: [https://www.geeksforgeeks.org/polymorphism-in-python/amp/]( https:/www.geeksforgeeks.org/polymorphism-in-python/amp/)