PetPals

Name : Ilakiya R

Create SQL Schema from the pet and user class, use the class attributes for table column names. 1. Create and implement the mentioned class and the structure in your application.

Name (string): The name of the pet.
Age (int): The age of the pet.
Breed (string): The breed of the pet. Methods:
Constructor to initialize Name, Age, and Breed.
Getters and setters for attributes.
ToString() method to provide a string representation of the pet.

```
class Pet:
   def __init__(self,name,age,breed):
       self._name = name
       self._age = age
       self._breed = breed
   @property
   def name(self):
       return self._name
   @name.setter
   def name(self,name):
       self._name = name
   @property
   def age(self):
       return self._age
   @age.setter
   def age(self,age):
       self._age = age
   @property
   def breed(self):
       return self._breed
   @breed.setter
   def breed(self,breed):
       self._breed = breed
   def to_string(self):
       return f"Name :{self._name},Age :{self._age},Breed :{self._breed}"

if __name__ == "__main__":
   pet1 = Pet("Bloo",5,"Golden Retriver")
   print(pet1.to_string())
   pet1.age = 4
```

```
    pet1.breed = "Beagle"
    print(pet1.to_string())
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe H:/pythonProject1/petpals.py
Name :Bloo,Age :5,Breed :Golden Retriver
Name :Bloo,Age :4,Breed :Beagle

Process finished with exit code 0
```

2. Dog Class (Inherits from Pet):
Additional Attributes:
DogBreed (string): The specific breed of the dog.
Additional Methods:
Constructor to initialize DogBreed.
Getters and setters for DogBreed.

```
class Pet:
    def __init__(self,name,age,breed):
        self._name = name
        self._age = age
        self._breed = breed
    @property
    def name(self):
        return self._name
    @name.setter
    def name(self,name):
        self._name = name
    @property
    def age(self):
        return self._age
    @age.setter
    def age(self,age):
        self._age = age
    @property
    def breed(self):
        return self._breed
    @breed.setter
    def breed(self,breed):
        self._breed = breed
    def to_string(self):
        return f"Name :{self._name},Age :{self._age},Breed :{self._breed}"

if __name__ == "__main__":
    pet1 = Pet("Bloo",5,"Golden Retriver")
    print(pet1.to_string())
```

```
    pet1.age = 4
    pet1.breed = "Beagle"
    print(pet1.to_string())

class Dog(Pet):
    def __init__(self, name, age, breed, dog_breed):
        super().__init__(name, age, breed)
        self._dog_breed = dog_breed

    @property
    def dog_breed(self):
        return self._dog_breed

    @dog_breed.setter
    def dog_breed(self, dog_breed):
        self._dog_breed = dog_breed
    def to_string(self):
        return "Name : {},Age : {},Breed :
{}".format(self._name,self._age,self._dog_breed)


class Cat(Pet):
    def __init__(self, name, age, breed, cat_color):
        super().__init__(name, age, breed)
        self._cat_color = cat_color

    @property
    def cat_color(self):
        return self._cat_color

    @cat_color.setter
    def cat_color(self, cat_color):
        self._cat_color = cat_color
    def to_string(self):
        return super().to_string() + f"Cat colour : {self._cat_color}"

dog1 = Dog("Bloo", 3, "Labrador", "Golden Retriever")
print(dog1.to_string())
cat1 = Cat("Sheero",3,"Siamese","Grey")
print(cat1.to_string())
```

```
Name : Bloo,Age : 3,Breed : Golden Retriever
Name :Sheero,Age :3,Breed :SiameseCat colour : Grey
```

3. PetShelter Class:

Attributes:

availablePets (List of Pet): A list to store available pets for adoption.
Methods:
AddPet(Pet pet): Adds a pet to the list of available pets.
RemovePet(Pet pet): Removes a pet from the list of available pets.
ListAvailablePets(): Lists all available pets in the shelter.

```
class Cat(Pet):
   def __init__(self, name, age, breed, cat_color):
       super().__init__(name, age, breed)
       self._cat_color = cat_color

   @property
   def cat_color(self):
       return self._cat_color

   @cat_color.setter
   def cat_color(self, cat_color):
       self._cat_color = cat_color
   def to_string(self):
       return super().to_string() + f"Cat colour : {self._cat_color}"

dog1 = Dog("Bloo", 3, "Labrador", "Golden Retriever")
print(dog1.to_string())
cat1 = Cat("Sheero",3,"Siamese","Grey")
print(cat1.to_string())


class PetShelter:
   def __init__(self):
       self.availablePets = []

   def add_pet(self, pet):
       self.availablePets.append(pet)

   def remove_pet(self, pet):
       if pet in self.availablePets:
           self.availablePets.remove(pet)
       else:
           print(f"{pet.get_name()} is not available for adoption.")

   def list_available_pets(self):
       print("Available Pets:")
       for pet in self.availablePets:
           print(pet.to_string())

shelter = PetShelter()
dog1 = Dog("Bloo", 3, "Labrador", "Golden Retriever")
cat1 = Cat("Sheero", 2, "Siamese", "Grey")
```

```
shelter.add_pet(dog1)
shelter.add_pet(cat1)
shelter.list_available_pets()
shelter.remove_pet(dog1)
print("After removing Bloo : ")
shelter.list_available_pets()
```

```
Available Pets:
Name : Bloo,Age : 3,Breed : Golden Retriever
Name :Sheero,Age :2,Breed :SiameseCat colour : Grey
After removing Bloo :
Available Pets:
Name :Sheero,Age :2,Breed :SiameseCat colour : Grey

Process finished with exit code 0
```

4. Donation Class (Abstract):
Attributes:
DonorName (string): The name of the donor.
Amount (decimal): The donation amount.
Methods:
Constructor to initialize DonorName and Amount.
Abstract method RecordDonation() to record the donation (to be implemented in derived classes).

CashDonation Class (Derived from Donation):
Additional Attributes:
DonationDate (DateTime): The date of the cash donation.
Additional Methods:
Constructor to initialize DonationDate.
Implementation of RecordDonation() to record a cash donation.

ItemDonation Class (Derived from Donation):
Additional Attributes:
ItemType (string): The type of item donated (e.g., food, toys).
Additional Methods:
Constructor to initialize ItemType.
Implementation of RecordDonation() to record an item donation.

```
from abc import ABC,abstractmethod
from datetime import datetime
```

```python
class Donation(ABC):
    def __init__(self, donor_name, amount):
        self.donor_name = donor_name
        self.amount = amount

    @abstractmethod
    def record_donation(self):
        pass

class CashDonation(Donation):
    def __init__(self, donor_name, amount, donation_date):
        super().__init__(donor_name, amount)
        self.donation_date = donation_date

    def record_donation(self):
        print(f"Cash donation of ${self.amount} recorded on {self.donation_date}
by {self.donor_name}.")


class ItemDonation(Donation):
    def __init__(self, donor_name, amount, item_type):
        super().__init__(donor_name, amount)
        self.item_type = item_type

    def record_donation(self):
        print(f"Item donation of {self.item_type} with a value of
${self.amount} recorded by {self.donor_name}.")
cash_donation = CashDonation("Ilakiya", 5430, datetime.now())
cash_donation.record_donation()
item_donation = ItemDonation("Rangaraju", 10786, "Food")
item_donation.record_donation()
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe H:/pythonProject1/2.py
Cash donation of $5430 recorded on 2024-05-02 10:29:12.293732 by Ilakiya.
Item donation of Food with a value of $10786 recorded by Rangaraju.

Process finished with exit code 0
```

5. IAdoptable Interface/Abstract Class:
Methods:
Adopt(): An abstract method to handle the adoption process.
AdoptionEvent Class:
Attributes:
Participants (List of IAdoptable): A list of participants (shelters and adopters) in the adoption event.

Methods:
HostEvent(): Hosts the adoption event.
RegisterParticipant(IAdoptable participant): Registers a participant for the event.

```
from abc import ABC, abstractmethod
class IAdoptable(ABC):
   @abstractmethod
   def adopt(self):
       pass
class AdoptionEvent:
   def __init__(self):
       self.participants = []
   def host_event(self):
       print("Adoption event is being hosted.")
       print("Participants who are in the event:")
       for participant in self.participants:
           print(participant.__class__.__name__)
   def register_participant(self, participant):
       self.participants.append(participant)

class Shelter(IAdoptable):
   def adopt(self):
       print("Adoption process handled by the shelter.")

class Adopter(IAdoptable):
   def adopt(self):
       print("Adoption process handled by the adopter.")

event = AdoptionEvent()
shelter = Shelter()
adopter = Adopter()
event.register_participant(shelter)
event.register_participant(adopter)
event.host_event()
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe H:/pythonProject1/3.py
Adoption event is being hosted.
Participants who are in the event:
Shelter
Adopter

Process finished with exit code 0
```

6. Exceptions handling
Create and implement the following exceptions in your application.

Invalid Pet Age Handling:
In the Pet Adoption Platform, when adding a new pet to a shelter, the age of the pet should be a positive integer. Write a program that prompts the user to input the age of a pet. Implement exception handling to ensure that the input is a positive integer. If the input is not valid, catch the exception and display an error message. If the input is valid, add the pet to the shelter.
Null Reference Exception Handling:
In the Pet Adoption Platform, when displaying the list of available pets in a shelter, it's important to handle situations where a pet's properties (e.g., Name, Age) might be null. Implement exception handling to catch null reference exceptions when accessing properties of pets in the shelter and display a message indicating that the information is missing.

```
class InvalidPetAgeError(Exception):
   pass
class NullReferenceError(Exception):
   pass
class Pet:
   def __init__(self, name, age, breed):
       self.name = name
       self.age = age
       self.breed = breed

class PetShelter:
   def __init__(self):
       self.available_pets = []

   def add_pet(self, pet):
       self.available_pets.append(pet)

   def list_available_pets(self):
       print("Available Pets:")
       for pet in self.available_pets:
           try:
               if pet.name is None or pet.age is None or pet.breed is None:
                   raise NullReferenceError("Pet information is incomplete.")
               print(f"Name: {pet.name}, Age: {pet.age}, Breed: {pet.breed}")
           except NullReferenceError as e:
               print(f"Error: {e}")

shelter = PetShelter()

try:
```

```python
    name = input("Enter the name of the pet: ")
    age = int(input("Enter the age of the pet: "))
    if age <= 0:
        raise InvalidPetAgeError("Age must be a positive integer.")
    breed = input("Enter the breed of the pet: ")
    new_pet = Pet(name, age, breed)
    shelter.add_pet(new_pet)
except ValueError:
    print("Error: Age must be a positive integer.")
except InvalidPetAgeError as e:
    print(f"Error: {e}")

shelter.list_available_pets()
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe H:/pythonProject1/4.py
Enter the name of the pet: Bloo
Enter the age of the pet: -7
Error: Age must be a positive integer.
Available Pets:

Process finished with exit code 0
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe H:/pythonProject1/4.py
Enter the name of the pet: Bloo
Enter the age of the pet: 3
Enter the breed of the pet: Beagle
Available Pets:
Name: Bloo, Age: 3, Breed: Beagle

Process finished with exit code 0
```

Insufficient Funds Exception:

Suppose the Pet Adoption Platform allows users to make cash donations to shelters. Write a program that prompts the user to enter the donation amount. Implement exception handling to catch situations where the donation amount is less than a minimum allowed amount (e.g., $10). If the donation amount is insufficient, catch the exception and display an error message. Otherwise, process the donation.

File Handling Exception:

In the Pet Adoption Platform, there might be scenarios where the program needs to read data from a file (e.g., a list of pets in a shelter). Write a program that attempts to read data from a file. Implement exception handling to catch any file-related exceptions (e.g.,

FileNotFoundException) and display an error message if the file is not found or cannot be read.

```
class InsufficientFundsError(Exception):
   pass

class FileHandlingError(Exception):
   pass

class PetShelter:
   def __init__(self):
       self.available_pets = []
       self.donation_funds = 0

   def add_pet(self, pet):
       self.available_pets.append(pet)

   def make_donation(self, amount):
       try:
           if amount < 10:
               raise InsufficientFundsError("Minimum donation amount is $10.")
           else:
               self.donation_funds += amount
               print(f"Donation of ${amount} processed successfully.")
       except InsufficientFundsError as e:
           print(f"Error: {e}")

   def read_from_file(self, filename):
       try:
           with open(filename, 'r') as f:
               for line in f:
                   print(line.strip())
               pass
       except FileNotFoundError:
           raise FileHandlingError(f"File '{filename}' not found.")
       except Exception as e:
           raise FileHandlingError(f"Error reading file: {e}")



shelter = PetShelter()
try:
   donation_amount = float(input("Enter the donation amount: $ "))
   shelter.make_donation(donation_amount)
except ValueError:
       print("Error: Invalid donation amount. Please enter a valid number.")
try:
   shelter.read_from_file("pet.txt")
```

```
    pass

except FileHandlingError as e:
    print(f"Error : {e}")
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/file handling.py"
Enter the donation amount: $ 988
Donation of $988.0 processed successfully.
Error : File 'pets.txt' not found.

Process finished with exit code 0
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/file handling.py"
Enter the donation amount: $ 6
Error: Minimum donation amount is $10.
Error : File 'pets.txt' not found.

Process finished with exit code 0
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/file handling.py"
Enter the donation amount: $ 789
Donation of $789.0 processed successfully.
Hi ilakiya!!! The required file is present here...

Process finished with exit code 0
```

Custom Exception for Adoption Errors:

Design a custom exception class called AdoptionException that inherits from Exception. In the Pet Adoption Platform, use this custom exception to handle adoption-related errors, such as attempting to adopt a pet that is not available or adopting a pet with missing information. Create instances of AdoptionException with different error messages and catch them appropriately in your program.

```
class AdoptionException(Exception):
    pass
class Pet:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed
class PetShelter:
    def __init__(self):
        self.available_pets = []
```

```
    def add_pet(self, pet):
        self.available_pets.append(pet)
    def adopt_pet(self, pet_name):
        try:
            for pet in self.available_pets:
                if pet.name == pet_name:
                    if pet.name is None or pet.age is None or pet.breed is None:
                        raise AdoptionException(f"Error adopting {pet_name}:
Missing information about the pet.")
                    else:
                        self.available_pets.remove(pet)
                        print(f"{pet_name} has been adopted!")
                        return
            raise AdoptionException(f"Error adopting {pet_name}: Pet not
available for adoption.")
        except AdoptionException as e:
            print(f"Adoption Error: {e}")
shelter = PetShelter()
shelter.add_pet(Pet("Juno", 3, "Labrador"))
shelter.add_pet(Pet("Sheero", 2, "Siamese"))
shelter.adopt_pet("Juno")
shelter.adopt_pet("Raju")
shelter.adopt_pet("Sheero")
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe H:/pythonProject1/5.py
Juno has been adopted!
Adoption Error: Error adopting Raju: Pet not available for adoption.
Sheero has been adopted!

Process finished with exit code 0
```

7. Database Connectivity

Create and implement the following tasks in your application.

Displaying Pet Listings:

Develop a program that connects to the database and retrieves a list of available pets from the "pets" table. Display this list to the user. Ensure that the program handles database connectivity exceptions gracefully, including cases where the database is unreachable.

```
import mysql.connector
from mysql.connector import Error

class Pet:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed
```

```python
def retrieve_pet_listings():

    con = mysql.connector.connect(
        host='localhost',
        user='root',
        password='root',
        database='emp'
    )

    if con.is_connected():
        print('Connected to MySQL database')

        cursor = con.cursor()
        cursor.execute("SELECT * FROM pets")
        pet_listings = cursor.fetchall()

        pets = []
        for row in pet_listings:
            pets.append(Pet(row[0], row[1], row[2]))
        return pets
pets = retrieve_pet_listings()
if pets:
    print("Available Pets:")
    for pet in pets:
        print(f"Name: {pet.name}, Age: {pet.age}, Breed: {pet.breed}")
else:
    print("No pets found.")
```

The table was not created

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe H:/pythonProject1/database.py
Connected to MySQL database
No pets found.

Process finished with exit code 0
```

After the table was created

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe H:/pythonProject1/database.py
Connected to MySQL database
Available Pets:
Name: Bloo, Age: 3, Breed: Labrador
Name: Juno, Age: 5, Breed: Golden retriver

Process finished with exit code 0
```

Donation Recording:

Create a program that records cash donations made by donors. Allow the user to input donor information and the donation amount and insert this data into the "donations" table in the database. Handle exceptions related to database operations, such as database errors or invalid inputs.

```
import mysql.connector
from mysql.connector import Error


class DatabaseConnectionError(Exception):
   pass

def record_donation(donor_name, donation_amount):
   con = mysql.connector.connect(
       host='localhost',
       user='root',
       password='root',
       database='emp'
   )

   if con.is_connected():
       print('Connected to MySQL database')

       cursor = con.cursor()
       cursor.execute("INSERT INTO donation (donor_name, donation_amount)
VALUES (%s, %s)",
                      (donor_name, donation_amount))
       con.commit()

       print("Donation recorded successfully!")
       cursor.close()
       con.close()
       print('Connection closed.')


donor_name = input("Enter donor name: ")
donation_amount = float(input("Enter donation amount: $ "))
record_donation(donor_name, donation_amount)
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/database 2.py"
Enter donor name: Ilakiya
Enter donation amount: $ 755
Connected to MySQL database
Donation recorded successfully!
Connection closed.

Process finished with exit code 0
```

Adoption Event Management:

Build a program that connects to the database and retrieves information about upcoming adoption events from the "adoption_events" table. Allow the user to register for an event by adding their details to the "participants" table. Ensure that the program handles database connectivity and insertion exceptions properly.

```
import mysql.connector
class DatabaseConnectionError(Exception):
   pass
def retrieve_upcoming_events():
   con = mysql.connector.connect(
       host='localhost',
       user='root',
       password='root',
       database='emp'
   )

   if con.is_connected():
       print('Connected to MySQL database')
       cursor = con.cursor()
       cursor.execute("SELECT * FROM adoption_events WHERE event_date >=
CURDATE()")
       upcoming_events = cursor.fetchall()

       events = []
       for row in upcoming_events:
           events.append({
               'event_id': row[0],
               'event_name': row[1],
               'event_date': row[2],
               'location': row[3]
           })

       cursor.close()
       con.close()
       print('Connection closed.')
       return events
def register_for_event(event_id, participant_name, participant_email=None,
participant_phone=None):
   conn = mysql.connector.connect(
       host='localhost',
       user='root',
       password='root',
       database='emp'
   )
   if conn.is_connected():
       print('Connected to MySQL database')
       cursor = conn.cursor()
```

```
        insert_query = "INSERT INTO participants (event_id, participant_name,
participant_email, participant_phone) VALUES (%s, %s, %s, %s)"
        participant_data = (event_id, participant_name, participant_email,
participant_phone)
        cursor.execute(insert_query, participant_data)
        conn.commit()
        cursor.close()
        conn.close()
        print('Connection closed.')
        print("Registration successful!")
events = retrieve_upcoming_events()
if events:
   print("Upcoming Adoption Events:")
   for event in events:
        print(f"Event ID: {event['event_id']}, Event Name:
{event['event_name']}, Date: {event['event_date']}, Location:
{event['location']}")
   else:
        print("No upcoming events found.")
event_id = int(input("Enter the event ID you want to register for: "))
participant_name = input("Enter your name: ")
participant_email = input("Enter your email (optional): ")
participant_phone = input("Enter your phone number (optional): ")
register_for_event(event_id, participant_name, participant_email,
participant_phone)
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\python.exe "H:/pythonProject1/database 3.py"
Connected to MySQL database
Connection closed.
Enter the event ID you want to register for: 1
Enter your name: Ilakiya
Enter your email (optional): ilakiya@gmail.com
Enter your phone number (optional): 9790342951
Connected to MySQL database
Connection closed.
Registration successful!

Process finished with exit code 0
```