CODING CHALLENGE

LOGESH D

- Create and implement the mentioned class and the structure in your application. Pet Class: Attributes: • 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 iname(self):
        return self._name
    @iname.setter
    def iname(self, name):
        self._name = name
    @property
    def iage(self):
        return self._age
    @iage.setter
    def iage(self, age):
        self._age = age
    @property
    def ibreed(self):
        return self._breed
    ibreed.setter
    def ibreed(self, breed):
        self._breed = breed
    def to_string(self):
       print("Name: {}, Age: {}, Breed: {}".format(self._name,
self._age, self._breed))

pet1 = Pet("Bobooo", 3, "Golden Retriever")
pet1.to_string()
pet1.iage=4
pet1.ibreed("Labrador")
pet1.to_string()
```

```
Name: Bobooo, Age: 3, Breed: Golden Retriever
Name: Bobooo, Age: 4, Breed: Labrador
```

- 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. Cat Class (Inherits from Pet): Additional Attributes: • CatColor (string): The color of the cat. Additional Methods: • Constructor to initialize CatColor. • Getters and setters for CatColor.

```
class Dog(Pet):
    def __init__(self, name, age, breed, dog_breed):
        super().__init__(name, age, breed)
        self._dog_breed = dog_breed
    @property
    def dogbreed(self):
        return self._dog_breed
    @dogbreed.setter
    def dogbreed(self, dog_breed):
        self._dog_breed = dog_breed

    def to_string(self):
        return "Name: {}, Age: {}, Breed: {}".format(self._name,
self._age, self._breed)


class Cat(Pet):
    def __init__(self, name, age, breed, cat_color):
        super().__init__(name, age, breed)
        self._cat_color = cat_color
    @property
    def catcolor(self):
        return self._cat_color
    @catcolor.setter
    def catcolor(self, cat_color):
        self._cat_color = cat_color

    def to_string(self):
        return super().to_string() + f", Cat Color:
{self._cat_color}"


dog1 = Dog("Boboo", 3, "Golden Retriever", "Labrador")
print(dog1.to_string())
cat1 = Cat("PAtlu", 2, "Siamese", "White")
print(cat1.to_string())
```

```
Name: Boboo, Age: 3, Breed: Golden Retriever
Name: PAtlu, Age: 2, Breed: Siamese
```

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

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

    def remove_pet(self, pet):
        if pet in self.available_pets:
            self.available_pets.remove(pet)
        else:
            print(f"{pet.get_name()} is not available in the
shelter.")

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



shelter = PetShelter()

dog1 = Dog("Buboo", 3, "Golden Retriever", "Labrador")
cat1 = Cat("Patlu", 2, "Siamese", "White")
shelter.add_pet(dog1)
shelter.add_pet(cat1)
shelter.list_available_pets()
shelter.remove_pet(dog1)
print("After removing Buboo:")
shelter.list_available_pets()
```

```
Available Pets:
Name: Buboo, Age: 3, Breed: Golden Retriever
After removing Buboo:
Available Pets:
```

- 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
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("Akil", 5000, datetime.now())
cash_donation.record_donation()
item_donation = ItemDonation("Kumar", 10000, "Food")
item_donation.record_donation()
```

```
Cash donation of $5000 recorded on 2024-05-02 09:31:52.599515 by Akil.
Item donation of Food with a value of $10000 recorded by Kumar.
```

- 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 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()
```

```
Adoption event is being hosted.
Participants in the event:
1 Shelter
1 Adopter
```

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

  • **Invalid Pet Age Handling**: o 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**: o 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:

    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()
```

```
Enter the name of the pet: Buttaa
Enter the age of the pet: -3
Error: Age must be a positive integer.
Available Pets:
```

```
Enter the name of the pet: Buboo
Enter the age of the pet: 3
Enter the breed of the pet: Golden Retriever
Available Pets:
Name: Buboo, Age: 3, Breed: Golden Retriever
```

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("codchall.txt")

    pass

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

```
Enter the donation amount: $ -5000
Error: Minimum donation amount is $10.
Error: File 'coding.txt' not found.
```

```
Enter the donation amount: $ 5000
Donation of $5000.0 processed successfully.
Error: File 'coding.txt' not found.
```

```
Enter the donation amount: $ 4500
Donation of $4500.0 processed successfully.
hi there file is present !!
```

Custom Exception for Adoption Errors: o 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("Butta", 3, "German Shepard"))
```

```
shelter.add_pet(Pet("Whiskers", 2, "Siamese"))
shelter.adopt_pet("Butta")
shelter.adopt_pet("Honey")
shelter.adopt_pet("Whiskers")
```

```
Butta has been adopted!
Adoption Error: Error adopting Honey: Pet not available for adoption.
Whiskers has been adopted!
```

```
Butta has been adopted!
Adoption Error: Error adopting Honey: Pet not available for adoption.
Whiskers has been adopted!
Bruno has been adopted!
```

- Database Connectivity Create and implement the following tasks in your application. • Displaying Pet Listings: o 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

def retrieve_pet_listings():

    conn = mysql.connector.connect(
        host='localhost',
        user='root',
        password='root',
        database='may2'
    )
```

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

        cursor = conn.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.")
```

pets table output:

```
Connected to MySQL database
Available Pets:
Name: Bruno, Age: 2, Breed: Pug
Name: Butta, Age: 3, Breed: Country
```

Table name: pet (doesn’t exist)

```
File "C:\Users\Sathish\AppData\Local\Programs\Python\pythonProject\Lib\site-packages\mysql\connector\cursor_cext.py", line 374, in execute
  result = self._cnx.cmd_query(
           ^^^^^^^^^^^^^^^^^^^
File "C:\Users\Sathish\AppData\Local\Programs\Python\pythonProject\Lib\site-packages\mysql\connector\opentelemetry\context_propagation.py", line 74, in wrapp
  return method(cnx, *args, **kwargs)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "C:\Users\Sathish\AppData\Local\Programs\Python\pythonProject\Lib\site-packages\mysql\connector\connection_cext.py", line 669, in cmd_query
  raise get_mysql_exception(
ysql.connector.errors.ProgrammingError: 1146 (42S02): Table 'may2.pet' doesn't exist
```

- Donation Recording: o 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):
    conn = mysql.connector.connect(
        host='localhost',
        user='root',
        password='root',
        database='may2'
    )

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

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

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


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

```
Enter donor name: logesh
Enter donation amount: $ 40000
Connected to MySQL database
Donation recorded successfully!
Connection closed.
```

- Adoption Event Management: o 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():
    conn = mysql.connector.connect(
        host='localhost',
        user='root',
        password='root',
        database='may2'
    )

    if conn.is_connected():
        print('Connected to MySQL database')
        cursor = conn.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()
        conn.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='may2'
    )

    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)
```

```
Connected to MySQL database
Connection closed.
Enter the event ID you want to register for: 201
Enter your name: Logesh
Enter your email (optional): logesh@gmail.com
Enter your phone number (optional): 9874521630
Connected to MySQL database
Connection closed.
Registration successful!

Process finished with exit code 0
```