Pet pals

**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

    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}, Breed: {self.breed}"
```

**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.**

```
from entity.model.pet import Pet

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

    def __str__(self):
        return super().__str__() + f", Dog Breed: {self.dog_breed}"
```

```
from entity.model.pet import Pet
```

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

    def __str__(self):
        return super().__str__() + f", Cat Color: {self.cat_color}"
```

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.

```
from dog import Dog
from cat import Cat


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.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("rocky", 3, "Labrador", "black")
cat1 = Cat("tommy", 2, "labrador", "white")

shelter.add_pet(dog1)
shelter.add_pet(cat1)
```

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

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.

```
# donation.py
from abc import ABC, abstractmethod

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}.")
```

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.

```
# adoption_event.py
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.")
```

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.

```
# exceptions.py
class InvalidPetAgeError(Exception):
    pass

class NullReferenceError(Exception):
    pass

class InsufficientFundsError(Exception):
    pass

class FileHandlingError(Exception):
    pass

class AdoptionException(Exception):
    pass
```

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 entity.model.pet import Pet

class PetRepository:
    def __init__(self, connection):
        self.connection = connection

    def retrieve_pet_listings(self):
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM pets")
            pet_listings = cursor.fetchall()
            pets = [Pet(*row) for row in pet_listings]
            return pets
        except mysql.connector.Error as e:
            print(f"Error retrieving pet listings: {e}")
            return []
```

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 entity.model.donation import Donation, CashDonation, ItemDonation

class DonationRepository:
    def __init__(self, connection):
        self.connection = connection

    def record_donation(self, donation):
        try:
            cursor = self.connection.cursor()
```

```
            if isinstance(donation, CashDonation):
                cursor.execute("INSERT INTO donations (donor_name, amount)
VALUES (%s, %s)", (donation.donor_name, donation.amount))
            elif isinstance(donation, ItemDonation):
                cursor.execute("INSERT INTO donations (donor_name, amount,
item_type) VALUES (%s, %s, %s)", (donation.donor_name, donation.amount,
donation.item_type))
            self.connection.commit()
            print("Donation recorded successfully!")
        except mysql.connector.Error as e:
            print(f"Error recording donation: {e}")
```

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 AdoptionEventRepository:
    def __init__(self, connection):
        self.connection = connection

    def retrieve_upcoming_events(self):
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM adoption_events WHERE event_date
>= CURDATE()")
            upcoming_events = cursor.fetchall()
            return upcoming_events
        except mysql.connector.Error as e:
            print(f"Error retrieving upcoming events: {e}")
            return []
```

Main.py

```
# main.py
import mysql.connector
from entity.database.pet_repository import PetRepository
```

```
from entity.database.donation_repository import DonationRepository
from entity.database.adoption_event_repository import
AdoptionEventRepository
from entity.model.pet import Pet
from entity.model.donation import CashDonation, ItemDonation
from entity.model.adoption_event import Shelter, Adopter, AdoptionEvent
from entity.exceptions import InvalidPetAgeError, NullReferenceError,
InsufficientFundsError, FileHandlingError, AdoptionException


try:
    connection = mysql.connector.connect(
        host='127.0.0.1',
        user='root',
        password="ani28790'",
        database='pet__adopt'
    )
    if connection.is_connected():
        print('Connected to MySQL database')
except mysql.connector.Error as e:
    print(f"Error connecting to MySQL database: {e}")


pet_repo = PetRepository(connection)
pets = pet_repo.retrieve_pet_listings()
if pets:
    print("Available Pets:")
    for pet in pets:
        print(pet)
else:
    print("No pets found.")


try:
    donor_name = input("Enter donor name: ")
    donation_amount = float(input("Enter donation amount: $ "))
    if donation_amount < 10:
        raise InsufficientFundsError("Minimum donation amount is $10.")
    donation_type = input("Enter donation type (cash/item): ")
    if donation_type.lower() == "cash":
        donation_date = input("Enter donation date (YYYY-MM-DD): ")
        donation = CashDonation(donor_name, donation_amount, donation_date)
    elif donation_type.lower() == "item":
        item_type = input("Enter item type: ")
        donation = ItemDonation(donor_name, donation_amount, item_type)
    else:
        raise ValueError("Invalid donation type")
```

```
    donation_repo = DonationRepository(connection)
    donation_repo.record_donation(donation)
except ValueError as e:
    print(f"Error: {e}")
except InsufficientFundsError as e:
    print(f"Error: {e}")
except mysql.connector.Error as e:
    print(f"Database error: {e}")


try:
    event_repo = AdoptionEventRepository(connection)
    upcoming_events = event_repo.retrieve_upcoming_events()
    if upcoming_events:
        print("Upcoming Adoption Events:")
        for event in upcoming_events:
            print(f"Event ID: {event['event_id']}, Event Name:
{event['event_name']}, Date: {event['event_date']}, Location:
{event['location']}")
        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): ")
        event = AdoptionEvent()
        event.register_participant(Shelter())
        event.register_participant(Adopter())
        event.host_event()
    else:
        print("No upcoming events found.")
except mysql.connector.Error as e:
    print(f"Database error: {e}")


try:
    if connection.is_connected():
        connection.close()
        print("MySQL connection closed")
except mysql.connector.Error as e:
    print(f"Error closing MySQL connection: {e}")
```