Coding Challenges: PetPals, The Pet Adoption Platform

Student Name: Sugandan Elangovan

Problem Statement:
PetPals, The Pet Adoption Platform scenario is a software system designed to facilitate the adoption of
pets, such as dogs and cats, from shelters or rescue organizations. This platform serves as a digital
marketplace where potential adopters can browse and select pets, shelters can list available pets, and
donors can contribute to support animal welfare

Implement OOPs

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.
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(IAdoptable):
    def __init__(self, pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id):
        self._pet_id = pet_id
        self._name = name
        self._age = age
        self._breed = breed
        self._pet_type = pet_type
        self._available_for_adoption = available_for_adoption
        self._shelter_name = shelter_name
        self._owner_id = owner_id
        self._shelter_id = shelter_id

    def get_pet_id(self):
        return self._pet_id

    def set_pet_id(self, pet_id):
        self._pet_id = pet_id

    def get_name(self):
        return self._name
```

```
def set_name(self, name):
    self._name = name

def get_age(self):
    return self._age

def set_age(self, age):
    self._age = age

def get_breed(self):
    return self._breed

def set_breed(self, breed):
    self._breed = breed

def get_pet_type(self):
    return self._pet_type

def set_pet_type(self, pet_type):
    self._pet_type = pet_type

def is_available_for_adoption(self):
    return self._available_for_adoption

def set_available_for_adoption(self, available_for_adoption):
    self._available_for_adoption = available_for_adoption

def get_shelter_name(self):
    return self._shelter_name

def set_shelter_name(self, shelter_name):
    self._shelter_name = shelter_name

def get_owner_id(self):
    return self._owner_id

def set_owner_id(self, owner_id):
    self._owner_id = owner_id

def get_shelter_id(self):
    return self._shelter_id

def set_shelter_id(self, shelter_id):
    self._shelter_id = shelter_id

  def Adopt(self):
    try:
```

```
            print(f"Adoption process handled for pet {self._name}")
        except Exception as e:
            raise AdoptionException(f"Error handling adoption: {e}")


    def __str__(self):
        try:
            return f"{self._name}, {self._age}, {self._breed}, {self._pet_type},
{self._available_for_adoption}, {self._shelter_name}, {self._owner_id}, {self._shelter_id}"
        except AttributeError:
            raise NullReferenceException("Pet information is missing.")
```

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, pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id, dog_breed):
        super().__init__(pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id)
        self._dog_breed = dog_breed

    def get_dog_breed(self):
        return self._dog_breed

    def set_dog_breed(self, dog_breed):
        self._dog_breed = dog_breed

class Cat(Pet):
    def __init__(self, pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id, cat_color):
        super().__init__(pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id)
        self._cat_color = cat_color

    def get_cat_color(self):
        return self._cat_color
```

```
    def set_cat_color(self, cat_color):
        self._cat_color = 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.

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

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

    def remove_pet(self, pet):
        self.available_pets.remove(pet)

    def list_available_pets(self):
        if not self.available_pets:
            print("No pets available for adoption.")
        else:
            print("Available Pets:")
            for pet in self.available_pets:
                try:
                    print(pet)
                except NullReferenceException as nre:
                    print(f"Error: {nre}")
                    continue
```

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).

```
class Donation(ABC):
    def __init__(self, donation_id, donor_name, donation_type, donation_amount, donation_item,
donation_date,
                 shelter_id):
        self.donation_id = donation_id
        self.donor_name = donor_name
```

```
        self.donation_type = donation_type
        self.donation_amount = donation_amount
        self.donation_item = donation_item
        self.donation_date = donation_date
        self.shelter_id = shelter_id

    @abstractmethod
    def record_donation(self):
        pass
```

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.

```
class CashDonation(Donation):
    def record_donation(self):
        try:
            cursor = db.conn.cursor()
            cursor.execute(
                "INSERT INTO Donations (DonationID,DonorName, DonationType, DonationAmount,
DonationItem, DonationDate, ShelterID) VALUES (?,?, ?, ?, ?, ?, ?)",
                (self.donation_id, self.donor_name, self.donation_type, self.donation_amount, None,
self.donation_date,
                 self.shelter_id))
            db.conn.commit()
            print(f"Cash donation of ${self.donation_amount} recorded on {self.donation_date} by
{self.donor_name}")
        except pyodbc.Error as ex:
            print(f"Error recording cash donation: {ex}")
```

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.

```
class ItemDonation(Donation):
    def record_donation(self):
        try:
            cursor = db.conn.cursor()
            cursor.execute(
                "INSERT INTO Donations (DonationID,DonorName, DonationType, DonationAmount,
DonationItem, DonationDate, ShelterID) VALUES (?,?, ?, ?, ?, ?, ?)",
                (self.donation_id, self.donor_name, self.donation_type, self.donation_amount,
self.donation_item,
```

```
            self.donation_date, self.shelter_id))
        db.conn.commit()
        print(f"Item donation of {self.donation_item} worth ${self.donation_amount} recorded
by {self.donor_name}")
    except pyodbc.Error as ex:
        print(f"Error recording item donation: {ex}")
```

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.

```
class IAdoptable(ABC):
    @abstractmethod
    def Adopt(self):
        pass

  def Adopt(self):
        try:

            print(f"Adoption process handled for pet {self._name}")
        except Exception as e:
            raise AdoptionException(f"Error handling adoption: {e}")

class AdoptionEvent:
    def __init__(self, event_id, event_name, event_date, location, city, organizer_id):
        self.event_id = event_id
        self.event_name = event_name
        self.event_date = event_date
        self.location = location
        self.city = city
        self.organizer_id = organizer_id

    def __str__(self):
        return f"Event ID: {self.event_id}, Name: {self.event_name}, Date: {self.event_date}, Location:
{self.location}"

    def HostEvent(self):
        print("Adoption event hosted successfully.")


class Participant:
```

```
    def __init__(self, participant_id, participant_name, participant_email, event_id, city):
        self.participant_id = participant_id
        self.participant_name = participant_name
        self.participant_email = participant_email
        self.event_id = event_id
        self.city = city

    def __str__(self):
        return f"Participant ID: {self.participant_id}, Name: {self.participant_name}, Email:
{self.participant_email}, Event ID: {self.event_id}, City: {self.city}"




    def add_participant(self, participant):
        self.participants_list.append(participant)

    def remove_participant(self, participant):
        self.participants_list.remove(participant)

    def list_participants(self):
        if not self.participants_list:
            print("No participants registered.")
        else:
            print("Registered Participants:")
            for participant in self.participants_list:
                try:
                    print(participant)
                except NullReferenceException as nre:
                    print(f"Error: {nre}")
                    continue
class PList:
    def __init__(self):
        self.participants_list = []

    @classmethod
    def create_instance(cls):
        return cls()

    def add_participant(self, participant):
        self.participants_list.append(participant)

    def remove_participant(self, participant):
        self.participants_list.remove(participant)

    def list_participants(self):
        if not self.participants_list:
            print("No participants registered.")
```

```
        else:
            print("Registered Participants:")
            for participant in self.participants_list:
                try:
                    print(participant)
                except NullReferenceException as nre:
                    print(f"Error: {nre}")
                    continue
```

6.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.

• Insufficient Funds Exception:
o 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:
o 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.

• 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 InvalidPetAgeException(Exception):
    pass


class FileHandlingException(Exception):
    pass


class NullReferenceException(Exception):
    pass


class DatabaseOperationException(Exception):
    Pass

class Participant:
    def __init__(self, participant_id, participant_name, participant_email, event_id, city):
        self.participant_id = participant_id
        self.participant_name = participant_name
        self.participant_email = participant_email
        self.event_id = event_id
        self.city = city

    def __str__(self):
        return f"Participant ID: {self.participant_id}, Name: {self.participant_name}, Email:
{self.participant_email}, Event ID: {self.event_id}, City: {self.city}"




    def add_participant(self, participant):
        self.participants_list.append(participant)

    def remove_participant(self, participant):
        self.participants_list.remove(participant)

    def list_participants(self):
        if not self.participants_list:
            print("No participants registered.")
        else:
            print("Registered Participants:")
            for participant in self.participants_list:
                try:
                    print(participant)
                except NullReferenceException as nre:
                    print(f"Error: {nre}")
```

```
                continue
class PList:
    def __init__(self):
        self.participants_list = []

    @classmethod
    def create_instance(cls):
        return cls()

    def add_participant(self, participant):
        self.participants_list.append(participant)

    def remove_participant(self, participant):
        self.participants_list.remove(participant)

    def list_participants(self):
        if not self.participants_list:
            print("No participants registered.")
        else:
            print("Registered Participants:")
            for participant in self.participants_list:
                try:
                    print(participant)
                except NullReferenceException as nre:
                    print(f"Error: {nre}")
                    continue

class Database:
    def __init__(self):
        self.conn = connect_to_sql_server()

    def get_available_pets(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM Pets")
            pets = cursor.fetchall()
            return pets
        except pyodbc.Error as ex:
            print(f"Error: {ex}")
            return []

    def get_upcoming_events(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM AdoptionEvents WHERE EventDate >= ?",
datetime.now())
            events = cursor.fetchall()
            return events
```

```
        except pyodbc.Error as ex:
            print(f"Error: {ex}")
            return []

    def register_participant(self, participant_id, participant_name, participant_email, event_id,
city):
        try:
            cursor = self.conn.cursor()
            cursor.execute(
                "INSERT INTO Participants (ParticipantID, ParticipantName,
ParticipantType,EventID,City) VALUES (?, ?, ?,?,?)",
                (participant_id, participant_name, participant_email, event_id, city))
            self.conn.commit()
            print("Participant registered successfully.")
        except pyodbc.Error as ex:
            print(f"Error registering participant: {ex}")
            raise DatabaseOperationException("Failed to register participant.")

    def retrieve_all_participants(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM Participants")
            pets = cursor.fetchall()
            return pets
        except pyodbc.Error as ex:
                print(f"Error: {ex}")
                return []


class CashDonation(Donation):
    def record_donation(self):
        try:
            cursor = db.conn.cursor()
            cursor.execute(
                "INSERT INTO Donations (DonationID,DonorName, DonationType, DonationAmount,
DonationItem, DonationDate, ShelterID) VALUES (?,?, ?, ?, ?, ?, ?)",
                (self.donation_id, self.donor_name, self.donation_type, self.donation_amount, None,
self.donation_date,
                 self.shelter_id))
            db.conn.commit()
            print(f"Cash donation of ${self.donation_amount} recorded on {self.donation_date} by
{self.donor_name}")
        except pyodbc.Error as ex:
            print(f"Error recording cash donation: {ex}")


class ItemDonation(Donation):
    def record_donation(self):
```

```
        try:
            cursor = db.conn.cursor()
            cursor.execute(
                "INSERT INTO Donations (DonationID,DonorName, DonationType, DonationAmount,
DonationItem, DonationDate, ShelterID) VALUES (?,?, ?, ?, ?, ?, ?)",
                (self.donation_id, self.donor_name, self.donation_type, self.donation_amount,
self.donation_item,
                 self.donation_date, self.shelter_id))
            db.conn.commit()
            print(f"Item donation of {self.donation_item} worth ${self.donation_amount} recorded
by {self.donor_name}")
        except pyodbc.Error as ex:
            print(f"Error recording item donation: {ex}")

def read_data_from_file(file_path):
    try:
        with open(file_path, 'r') as file:
            data = file.read()
            return data
    except FileNotFoundError:
        raise FileHandlingException("File not found.")
    except IOError:
        raise FileHandlingException("Error reading file.")


if __name__ == "__main__":

    db = Database()

    while True:
        display_menu()
        choice = input("Enter your choice: ")

        if choice == "1":

            shelter = PetShelter()

            pets = db.get_available_pets()
            for pet in pets:
                shelter.add_pet(Pet(*pet))

            shelter.list_available_pets()
        elif choice == "2":
            try:
                donation_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                donation_id = input("Enter donation ID: ")
                donor_name = input("Enter donor name: ")
                donation_amount = float(input("Enter donation amount: "))
```

```
            cash_donation = CashDonation(donation_id, donor_name, "Cash", donation_amount,
None, donation_date, 1)
            cash_donation.record_donation()
        except ValueError as ve:
            print(f"Error: {ve}")
    elif choice == "3":
        try:
            donation_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            donation_id = input("Enter donation ID: ")
            donor_name = input("Enter donor name: ")
            donation_amount = float(input("Enter donation amount: "))
            donation_item = input("Enter donation item: ")
            item_donation = ItemDonation(donation_id, donor_name, "Item", donation_amount,
donation_item,
                                         donation_date, 1)
            item_donation.record_donation()
        except ValueError as ve:
            print(f"Error: {ve}")
    elif choice == "4":

        try:
            events = db.get_upcoming_events()
            event_manager = AdoptionEventManager()
            for event in events:
                event_manager.add_event(AdoptionEvent(*event))
            event_manager.list_events()
        except DatabaseOperationException as doe:
            print(f"Database Operation Error: {doe}")
    elif choice == "5":
        try:
            participant_id = input("Enter your ID : ")
            event_id = input("Enter event ID to register for: ")
            participant_name = input("Enter your name: ")
            participant_email = input("Enter your email: ")
            city = input("Enter event City: ")
            db.register_participant(participant_id, participant_name, participant_email, event_id,
city)
        except DatabaseOperationException as doe:
            print(f"Database Operation Error: {doe}")
    elif choice == "6":
        try:
            file_name = input("Enter the file name: ")
            file_path = os.path.join(os.getcwd(), file_name)
            data = read_data_from_file(file_path)
            print("Data read successfully:")
            print(data)
        except FileHandlingException as fe:
            print(f"File Handling Error: {fe}")
```

```
        elif choice == "7":
            participants = PList()


            available_participants = db.retrieve_all_participants()

            for participant in available_participants:
                participants.add_participant(Participant(*participant))

            participants.list_participants()

        elif choice == "8":
            break
        else:
            print("Invalid choice. Please try again.")

    close_connection(db.conn)
```

7.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.

• 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.

• 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 pyodbc
from abc import ABC, abstractmethod
from datetime import datetime
import os
import dao
import exception
import entity

class Database:
    def __init__(self):
        self.conn = dao.connect_to_sql_server()
```

```
    def get_available_pets(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM Pets")
            pets = cursor.fetchall()
            return pets
        except pyodbc.Error as ex:
            print(f"Error: {ex}")
            return []

    def get_upcoming_events(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM AdoptionEvents WHERE EventDate >= ?",
datetime.now())
            events = cursor.fetchall()
            return events
        except pyodbc.Error as ex:
            print(f"Error: {ex}")
            return []

    def register_participant(self, participant_id, participant_name, participant_email, event_id,
city):
        try:
            cursor = self.conn.cursor()
            cursor.execute(
                "INSERT INTO Participants (ParticipantID, ParticipantName,
ParticipantType,EventID,City) VALUES (?, ?, ?,?,?)",
                (participant_id, participant_name, participant_email, event_id, city))
            self.conn.commit()
            print("Participant registered successfully.")
        except pyodbc.Error as ex:
            print(f"Error registering participant: {ex}")
            raise exception.DatabaseOperationException("Failed to register participant.")

    def retrieve_all_participants(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM Participants")
            pets = cursor.fetchall()
            return pets
        except pyodbc.Error as ex:
                print(f"Error: {ex}")
                return []
```

```
class CashDonation(entity.Donation):


    def record_donation(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute(
                "INSERT INTO Donations (DonationID,DonorName, DonationType, DonationAmount,
DonationItem, DonationDate, ShelterID) VALUES (?,?, ?, ?, ?, ?, ?)",
                (self.donation_id, self.donor_name, self.donation_type, self.donation_amount, None,
self.donation_date,
                 self.shelter_id))
            self.conn.commit()
            print(f"Cash donation of ${self.donation_amount} recorded on {self.donation_date} by
{self.donor_name}")
        except pyodbc.Error as ex:
            print(f"Error recording cash donation: {ex}")


class ItemDonation(entity.Donation):
    def record_donation(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute(
                "INSERT INTO Donations (DonationID,DonorName, DonationType, DonationAmount,
DonationItem, DonationDate, ShelterID) VALUES (?,?, ?, ?, ?, ?, ?)",
                (self.donation_id, self.donor_name, self.donation_type, self.donation_amount,
self.donation_item,
                 self.donation_date, self.shelter_id))
            self.conn.commit()
            print(f"Item donation of {self.donation_item} worth ${self.donation_amount} recorded
by {self.donor_name}")
        except pyodbc.Error as ex:
            print(f"Error recording item donation: {ex}")
```

Main.py

```
import pyodbc
from abc import ABC, abstractmethod
from datetime import datetime
import os
import dao
import exception
import entity
import util
```

```
def display_menu():
    print("1. List Available Pets")
    print("2. Record Cash Donation")
    print("3. Record Item Donation")
    print("4. List Upcoming Adoption Events")
    print("5. Register for an Adoption Event")
    print("6. Read Data from File")
    print("7. Read all Participants")
    print("8. Exit")


def read_data_from_file(file_path):
    try:
        with open(file_path, 'r') as file:
            data = file.read()
            return data
    except FileNotFoundError:
        raise exception.FileHandlingException("File not found.")
    except IOError:
        raise exception.FileHandlingException("Error reading file.")


if __name__ == "__main__":

    db = util.Database()

    while True:
        display_menu()
        choice = input("Enter your choice: ")

        if choice == "1":

            shelter = entity.PetShelter()

            pets = db.get_available_pets()
            for pet in pets:
                shelter.add_pet(entity.Pet(*pet))

            shelter.list_available_pets()
        elif choice == "2":
            try:
                donation_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                donation_id = input("Enter donation ID: ")
                donor_name = input("Enter donor name: ")
                donation_amount = float(input("Enter donation amount: "))
```

```
                cash_donation = util.CashDonation(donation_id, donor_name, "Cash",
donation_amount, None, donation_date, 1)
                cash_donation.record_donation()
            except ValueError as ve:
                print(f"Error: {ve}")
        elif choice == "3":
            try:
                donation_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                donation_id = input("Enter donation ID: ")
                donor_name = input("Enter donor name: ")
                donation_amount = float(input("Enter donation amount: "))
                donation_item = input("Enter donation item: ")
                item_donation = util.ItemDonation(donation_id, donor_name, "Item",
donation_amount, donation_item,
                                                  donation_date, 1)
                item_donation.record_donation()
            except ValueError as ve:
                print(f"Error: {ve}")
        elif choice == "4":

            try:
                events = db.get_upcoming_events()
                event_manager = entity.AdoptionEventManager()
                for event in events:
                    event_manager.add_event(entity.AdoptionEvent(*event))
                event_manager.list_events()
            except exception.DatabaseOperationException as doe:
                print(f"Database Operation Error: {doe}")
        elif choice == "5":
            try:
                participant_id = input("Enter your ID : ")
                event_id = input("Enter event ID to register for: ")
                participant_name = input("Enter your name: ")
                participant_email = input("Enter your email: ")
                city = input("Enter event City: ")
                db.register_participant(participant_id, participant_name, participant_email, event_id,
city)
            except exception.DatabaseOperationException as doe:
                print(f"Database Operation Error: {doe}")
        elif choice == "6":
            try:
                file_name = input("Enter the file name: ")
                file_path = os.path.join(os.getcwd(), file_name)
                data = read_data_from_file(file_path)
                print("Data read successfully:")
                print(data)
            except exception.FileHandlingException as fe:
                print(f"File Handling Error: {fe}")
```

```
        elif choice == "7":
            participants = entity.PList()


            available_participants = db.retrieve_all_participants()

            for participant in available_participants:
                participants.add_participant(entity.Participant(*participant))

            participants.list_participants()

        elif choice == "8":
            break
        else:
            print("Invalid choice. Please try again.")

    dao.close_connection(db.conn)
```

Util.py

```
import pyodbc
from abc import ABC, abstractmethod
from datetime import datetime
import os
import dao
import exception
import entity

class Database:
    def __init__(self):
        self.conn = dao.connect_to_sql_server()

    def get_available_pets(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM Pets")
            pets = cursor.fetchall()
            return pets
        except pyodbc.Error as ex:
            print(f"Error: {ex}")
            return []

    def get_upcoming_events(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM AdoptionEvents WHERE EventDate >= ?",
datetime.now())
            events = cursor.fetchall()
```

```
            return events
        except pyodbc.Error as ex:
            print(f"Error: {ex}")
            return []

    def register_participant(self, participant_id, participant_name, participant_email, event_id,
city):
        try:
            cursor = self.conn.cursor()
            cursor.execute(
                "INSERT INTO Participants (ParticipantID, ParticipantName,
ParticipantType,EventID,City) VALUES (?, ?, ?,?,?)",
                (participant_id, participant_name, participant_email, event_id, city))
            self.conn.commit()
            print("Participant registered successfully.")
        except pyodbc.Error as ex:
            print(f"Error registering participant: {ex}")
            raise exception.DatabaseOperationException("Failed to register participant.")

    def retrieve_all_participants(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM Participants")
            pets = cursor.fetchall()
            return pets
        except pyodbc.Error as ex:
                print(f"Error: {ex}")
                return []




class CashDonation(entity.Donation):


    def record_donation(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute(
                "INSERT INTO Donations (DonationID,DonorName, DonationType, DonationAmount,
DonationItem, DonationDate, ShelterID) VALUES (?,?, ?, ?, ?, ?, ?)",
                (self.donation_id, self.donor_name, self.donation_type, self.donation_amount, None,
self.donation_date,
                 self.shelter_id))
            self.conn.commit()
            print(f"Cash donation of ${self.donation_amount} recorded on {self.donation_date} by
{self.donor_name}")
```

```
        except pyodbc.Error as ex:
            print(f"Error recording cash donation: {ex}")


class ItemDonation(entity.Donation):
    def record_donation(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute(
                "INSERT INTO Donations (DonationID,DonorName, DonationType, DonationAmount,
DonationItem, DonationDate, ShelterID) VALUES (?,?, ?, ?, ?, ?, ?)",
                (self.donation_id, self.donor_name, self.donation_type, self.donation_amount,
self.donation_item,
                 self.donation_date, self.shelter_id))
            self.conn.commit()
            print(f"Item donation of {self.donation_item} worth ${self.donation_amount} recorded
by {self.donor_name}")
        except pyodbc.Error as ex:
            print(f"Error recording item donation: {ex}")
```

Entity.py

```
from abc import ABC, abstractmethod

import exception

class Donation(ABC):
    def __init__(self, donation_id, donor_name, donation_type, donation_amount, donation_item,
donation_date,
                 shelter_id):
        self.donation_id = donation_id
        self.donor_name = donor_name
        self.donation_type = donation_type
        self.donation_amount = donation_amount
        self.donation_item = donation_item
        self.donation_date = donation_date
        self.shelter_id = shelter_id

    @abstractmethod
    def record_donation(self):
        pass
class Pet:
    def __init__(self, pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id):
        self.pet_id = pet_id
        self.name = name
        self.age = age
```

```
    self.breed = breed
    self.pet_type = pet_type
    self.available_for_adoption = available_for_adoption
    self.shelter_name = shelter_name
    self.owner_id = owner_id
    self.shelter_id = shelter_id

def get_pet_id(self):
    return self._pet_id

def set_pet_id(self, pet_id):
    self._pet_id = pet_id

def get_name(self):
    return self._name

def set_name(self, name):
    self._name = name

def get_age(self):
    return self._age

def set_age(self, age):
    self._age = age

def get_breed(self):
    return self._breed

def set_breed(self, breed):
    self._breed = breed

def get_pet_type(self):
    return self._pet_type

def set_pet_type(self, pet_type):
    self._pet_type = pet_type

def is_available_for_adoption(self):
    return self._available_for_adoption

def set_available_for_adoption(self, available_for_adoption):
    self._available_for_adoption = available_for_adoption

def get_shelter_name(self):
    return self._shelter_name

def set_shelter_name(self, shelter_name):
    self._shelter_name = shelter_name
```

```
    def get_owner_id(self):
        return self._owner_id

    def set_owner_id(self, owner_id):
        self._owner_id = owner_id

    def get_shelter_id(self):
        return self._shelter_id

    def set_shelter_id(self, shelter_id):
        self._shelter_id = shelter_id

    def Adopt(self):
        try:
            # Implement adoption process
            print(f"Adoption process handled for pet {self._name}")
        except Exception as e:
            raise exception.AdoptionException(f"Error handling adoption: {e}")

    def __str__(self):
        try:
            return f"{self.name}, {self.age}, {self.breed}, {self.pet_type}, {self.available_for_adoption},
{self.shelter_name}, {self.owner_id}, {self.shelter_id}"
        except AttributeError:
            raise exception.NullReferenceException("Pet information is missing.")


class Dog(Pet):
    def __init__(self, pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id,
                 dog_breed):
        super().__init__(pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id)
        self.dog_breed = dog_breed

    def get_dog_breed(self):
        return self.dog_breed

    def set_dog_breed(self, dog_breed):
        self.dog_breed = dog_breed


class Cat(Pet):
    def __init__(self, pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id,
                 cat_color):
```

```
        super().__init__(pet_id, name, age, breed, pet_type, available_for_adoption, shelter_name,
owner_id, shelter_id)
        self.cat_color = cat_color

    def get_cat_color(self):
        return self.cat_color

    def set_cat_color(self, cat_color):
        self.cat_color = cat_color


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

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

    def remove_pet(self, pet):
        self.available_pets.remove(pet)

    def list_available_pets(self):
        if not self.available_pets:
            print("No pets available for adoption.")
        else:
            print("Available Pets:")
            for pet in self.available_pets:
                try:
                    print(pet)
                except exception.NullReferenceException as nre:
                    print(f"Error: {nre}")
                    continue


class AdoptionEvent:
    def __init__(self, event_id, event_name, event_date, location, city, organizer_id):
        self.event_id = event_id
        self.event_name = event_name
        self.event_date = event_date
        self.location = location
        self.city = city
        self.organizer_id = organizer_id

    def __str__(self):
        return f"Event ID: {self.event_id}, Name: {self.event_name}, Date: {self.event_date}, Location:
{self.location}"

    def HostEvent(self):
```

```
        print("Adoption event hosted successfully.")

class AdoptionEventManager:
    def __init__(self):
        self.events = []

    def add_event(self, event):
        self.events.append(event)

    def list_events(self):
        if not self.events:
            print("No upcoming adoption events.")
        else:
            print("Upcoming Adoption Events:")
            for event in self.events:
                print(event)

class Participant:
    def __init__(self, participant_id, participant_name, participant_email, event_id, city):
        self.participant_id = participant_id
        self.participant_name = participant_name
        self.participant_email = participant_email
        self.event_id = event_id
        self.city = city

    def __str__(self):
        return f"Participant ID: {self.participant_id}, Name: {self.participant_name}, Email:
{self.participant_email}, Event ID: {self.event_id}, City: {self.city}"




    def add_participant(self, participant):
        self.participants_list.append(participant)

    def remove_participant(self, participant):
        self.participants_list.remove(participant)

    def list_participants(self):
        if not self.participants_list:
            print("No participants registered.")
        else:
            print("Registered Participants:")
            for participant in self.participants_list:
                try:
                    print(participant)
                except exception.NullReferenceException as nre:
                    print(f"Error: {nre}")
```

```
                continue
class PList:
    def __init__(self):
        self.participants_list = []

    @classmethod
    def create_instance(cls):
        return cls()

    def add_participant(self, participant):
        self.participants_list.append(participant)

    def remove_participant(self, participant):
        self.participants_list.remove(participant)

    def list_participants(self):
        if not self.participants_list:
            print("No participants registered.")
        else:
            print("Registered Participants:")
            for participant in self.participants_list:
                try:
                    print(participant)
                except exception.NullReferenceException as nre:
                    print(f"Error: {nre}")
                    continue
class Donation(ABC):
    def __init__(self, donation_id, donor_name, donation_type, donation_amount, donation_item,
donation_date,
                 shelter_id):
        self.donation_id = donation_id
        self.donor_name = donor_name
        self.donation_type = donation_type
        self.donation_amount = donation_amount
        self.donation_item = donation_item
        self.donation_date = donation_date
        self.shelter_id = shelter_id

    @abstractmethod
    def record_donation(self):
        pass
```

Dao.py

```
import pyodbc


def connect_to_sql_server():
```

```
    try:
        conn = pyodbc.connect('Driver={SQL Server};'
                              'Server=DESKTOP-A08GADU\SQLEXPRESS01;'
                              'Database=PetPals;'
                              'Trusted_Connection=yes;')
        print("Connected Successfully")
        return conn
    except pyodbc.Error as ex:
        print(f"Error: {ex}")


def close_connection(conn):
    conn.close()
    print("Connection closed.")
```

Exception.py

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

    class AdoptionException(Exception):
        pass


    class InvalidPetAgeException(Exception):
        pass


    class FileHandlingException(Exception):
        pass


    class NullReferenceException(Exception):
        pass


    class DatabaseOperationException(Exception):
        pass

    class IAdoptable(ABC):
        @abstractmethod
        def Adopt(self):
            pass
```

OUTPUT:

```
"C:\Users\Sugandan\PycharmProjects\pythonProject\Hexaware foundation\Scripts\p
Connected Successfully
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
Enter your choice: 1
Available Pets:
Charlie, 2, Labrador Retriever, Dog, True, Chennai Pet Shelter, None, 1
Whiskers, 1, Siamese, Cat, True, Coimbatore Animal Care, None, 2
Rocky, 3, German Shepherd, Dog, True, Madurai Paws Haven, None, 3
Mittens, 2, Persian, Cat, True, Trichy Furry Friends, None, 4
Buddy, 1, Golden Retriever, Dog, True, Salem Animal Sanctuary, None, 5
Fluffy, 2, Ragdoll, Cat, True, Vellore Pet Haven, None, 6
Goldy, 2, Labrador Retriever, Dog, True, Chennai Pet Shelter, None, 1
Fluffs, 1, Siamese, Cat, True, Coimbatore Animal Care, None, 2
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
Enter your choice: 2
Enter donation ID: 108
Enter donor name: Ramu
Enter donation amount: 2000
Cash donation of $2000.0 recorded on 2024-03-18 12:15:46 by Ramu
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
```

```
Enter your choice: 3
Enter donation ID: 109
Enter donor name: Raju
Enter donation amount: 230
Enter donation item: Toys
Item donation of Toys worth $230.0 recorded by Raju
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
Enter your choice: 4
Upcoming Adoption Events:
Event ID: 2, Name: Furry Friends Fiesta, Date: 2024-03-20 11:30:00, Location: Race Course
Event ID: 3, Name: Paws Parade, Date: 2024-03-25 13:45:00, Location: Goripalayam Ground
Event ID: 4, Name: Trichy Pet Carnival, Date: 2024-04-02 10:00:00, Location: Maris Theater Ground
Event ID: 5, Name: Salem Pet Fest, Date: 2024-04-10 15:15:00, Location: Anna Park
Event ID: 6, Name: Vellore Adoption Drive, Date: 2024-04-18 12:30:00, Location: VIT University Ground
```

```
"C:\Users\Sugandan\PycharmProjects\pythonProject\Hexaware
Connected Successfully
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
Enter your choice: 5
Enter your ID : 110
Enter event ID to register for: 2
Enter your name: Kumar
Enter your email: Kumar@123.com
Enter event City: Chennai
Participant registered successfully.
```

```
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
Enter your choice: Employees.txt
Invalid choice. Please try again.
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
```

```
File Handling Error: File not found.
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
Enter your choice: 6
Enter the file name: employees.txt
Data read successfully:
John
4495
EEE
Ram
5567
IT
```

```
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
Enter your choice: 7
Registered Participants:
Participant ID: 1, Name: Aruna Nair, Email: Volunteer, Event ID: 1, City: Chennai
Participant ID: 2, Name: Karthik Raj, Email: Adopter, Event ID: 2, City: Coimbatore
Participant ID: 3, Name: Meera Devi, Email: Volunteer, Event ID: 3, City: Madurai
Participant ID: 4, Name: Vijay Kumar, Email: Adopter, Event ID: 4, City: Trichy
Participant ID: 5, Name: Priya Reddy, Email: Volunteer, Event ID: 5, City: Salem
Participant ID: 6, Name: Gopal Krishnan, Email: Adopter, Event ID: 6, City: Vellore
Participant ID: 106, Name: Mano, Email: manoman@gmail.com, Event ID: 2, City: Chennai
Participant ID: 110, Name: Kumar, Email: Kumar@123.com, Event ID: 2, City: Chennai
```

```
1. List Available Pets
2. Record Cash Donation
3. Record Item Donation
4. List Upcoming Adoption Events
5. Register for an Adoption Event
6. Read Data from File
7. Read all Participants
8. Exit
Enter your choice: 8
Connection closed.

Process finished with exit code 0
```