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.

Final Program:

import pyodbc

from abc import ABC, abstractmethod

from datetime import datetime

import os

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

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.")

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:

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.")

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 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 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 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 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 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 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)

**OUTPUT:**















