CODING CHALLENGE

NAME: ANISH
TOPIC : PET PALS

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.

```
from exception.invalidageerror import InvalidAgeError

class Pet:
    def __init__(self, name, age, breed):
        if not isinstance(age, int) or age < 0:
            raise InvalidAgeError("Age must be a non-negative integer")
        self.name = name
        self.age = age
        self.breed = breed

    def get_name(self):
        return self.name

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

    def get_age(self):
        return self.age
```

```python
    def set_age(self, age):
        if not isinstance(age, int) or age < 0:
            raise InvalidAgeError("Age must be a non-negative integer")
        self.age = age

    def get_breed(self):
        return self.breed

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

    def update_by_name(self, new_age=None, new_breed=None):
        if new_age is not None:
            if not isinstance(new_age, int) or new_age < 0:
                raise InvalidAgeError("Age must be a non-negative integer")
            self.age = new_age
        if new_breed is not None:
            self.breed = new_breed

    def __str__(self):
        return f"{self.name}, {self.age} years old, {self.breed}"
```

```
PetPals: The Pet Adoption Platform
1. Add a pet to the shelter
2. List available pets in the shelter
3. Record a cash donation
4. View donation data
5. Host an adoption event
6. Register for an adoption event
7. View adoption data
8. Adopt a pet
9. Exit
Enter your choice (1-9): 1
Enter pet type (dog/cat): dog
Enter pet name: tiger
Enter pet age: 4
Enter pet breed: labrador
--Database is Connected--
pet added to the shelter.
```

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.

```
from entity.pet import Pet

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

    def get_dog_breed(self):
        return self.dog_breed

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

    def __str__(self):
        return f"{self.name}, {self.age} years old, belongs to {self.breed},
{self.dog_breed}."
```

```
from entity.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 set_cat_color(self, cat_color):
        self.cat_color = cat_color

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

```
    def __str__(self):
        return f"{self.name}, {self.age} years old, belongs to {self.breed}, is of
{self.cat_color} in color"
```

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 exception.duplicateobjeerror import DuplicateObjError
from entity.pet import Pet

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

    def add_pet(self, pet):
        if not isinstance(pet, Pet):
            raise ValueError("Invalid pet type.")


        for existing_pet in self.available_pets:
            if (
                existing_pet.get_name() == pet.get_name()
                and existing_pet.get_age() == pet.get_age()
                and existing_pet.get_breed() == pet.get_breed()
            ):
                raise DuplicateObjError()

        self.available_pets.append(pet)
        print("pet added to the shelter.")

    def remove_pet(self, pet):
        if pet in self.available_pets:
            self.available_pets.remove(pet)
```

```
            # print("pet removed from the shelter.")

    def list_available_pets(self):
        if not self.available_pets:
            print("No pets available in the shelter.")
        else:
            print("Available Pets:")
            for pet in self.available_pets:
                print(pet)
```

```
Enter pet ID: 15
--Database is Connected--
Enter participant ID: 10
--Database is Connected--
--Database is Connected--
Adoption data inserted successfully.
Emma Brown adopted tommy successfully
--Database is Connected--
Pet is successfully removed from the shelter!
```

Donation Class (Abstract): Attributes:
• DonorName (string): The name of the donor.
• Amount (decimal): The donation amount.
Methods: • Constructor to initialize DonorName and Amount.
• Abstract method RecordDonation() to record the donation (to be implemented in derived classes).
CashDonation Class (Derived from Donation): Additional Attributes: • DonationDate (DateTime): The date of the cash donation.
Additional Methods: • Constructor to initialize DonationDate.
• Implementation of RecordDonation() to record a cash donation.
ItemDonation Class (Derived from Donation): Additional Attributes: • ItemType (string): The type of item donated (e.g., food, toys).
Additional Methods: • Constructor to initialize ItemType. • Implementation of RecordDonation() to record an item donation.

```
from abc import ABC, abstractmethod

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

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

```
from entity.donation import Donation

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} recorded.")
```

```
PetPals: The Pet Adoption Platform
1. Add a pet to the shelter
2. List available pets in the shelter
3. Record a cash donation
4. View donation data
5. Host an adoption event
6. Register for an adoption event
7. View adoption data
8. Adopt a pet
9. Exit
Enter your choice (1-9): 3
Enter donor name: nadhin
Enter donation amount: 102
--Database is Connected--
Cash donation recorded successfully.
```

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.

```python
from abc import ABC, abstractmethod

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

```python
from dao.interface.iadoptable import IAdoptable
from exception.invalidnameerror import InvalidNameError
from exception.adoptionerror import AdoptionException
from exception.filehandlingerror import FileHandlingException
import mysql.connector as sql
from util.dbconnutil import DBConnection
from dao.petshelter import PetShelterDAO

class AdoptionEventDAO(DBConnection, IAdoptable):
    def __init__(self):
        self.pet_shelter_dao = PetShelterDAO()

    def create_event(self):
        try:
            self.open_connection()
            create_event_query = '''
                CREATE TABLE IF NOT EXISTS Event (
                    ID INT PRIMARY KEY AUTO_INCREMENT,
                    Details VARCHAR(255) NOT NULL
                );
            '''
            self.stmt.execute(create_event_query)

            # self.close_connection()
        except Exception as e:
            print(f"Error creating Event table: {e}")

    def create_participants(self):
        try:
            self.open_connection()
            create_participants_query = '''
                CREATE TABLE IF NOT EXISTS Participants (
                    ID INT PRIMARY KEY AUTO_INCREMENT,
                    Name VARCHAR(255) NOT NULL
```

```
            );
        '''
        self.stmt.execute(create_participants_query)

        # self.close_connection()
    except Exception as e:
        print(f"Error creating Participants table: {e}")

def create_adoption(self):
    try:
        self.open_connection()
        create_adoption_query = '''
            CREATE TABLE IF NOT EXISTS Adopt (
                petname VARCHAR(50),
                petage INTEGER,
                petbreed VARCHAR(50),
                name VARCHAR(50)
            );
        '''
        self.stmt.execute(create_adoption_query)

        # self.close_connection()
    except Exception as e:
        print(f"Error creating Adopt table: {e}")

def register_participant(self):
    try:
        participant_name = input("Enter participant name: ")
        if not isinstance(participant_name, str):
            raise InvalidNameError()
        for char in participant_name:
            if not char.isalpha() and not char.isspace():
                raise InvalidNameError()
        self.open_connection()
        insert_query = "INSERT INTO Participants (Name) VALUES (%s)"
        self.stmt.execute(insert_query, (participant_name,))
        self.conn.commit()
        print(f"Participant '{participant_name}' added successfully.")
        # self.close_connection()
    except Exception as e:
        print(f"Error adding participant: {e}")
```

```
    def host_event(self):
        try:
            event_details = input("Enter event details: ")
            self.open_connection()
            insert_query = "INSERT INTO Event (Details) VALUES (%s)"
            self.stmt.execute(insert_query, (event_details,))
            self.conn.commit()
            print("Event hosted successfully.")
            # self.close_connection()
        except Exception as e:
            print(f"Error hosting event: {e}")

    def view_adoption(self):
        try:
            self.open_connection()
            view_adoption_query = "SELECT * FROM Adopt;"
            self.stmt.execute(view_adoption_query)
            result = self.stmt.fetchall()
            if result:
                print("Adoption table data:")
                for row in result:
                    print(row)
            else:
                raise FileHandlingException()
            # self.close_connection()
        except FileHandlingException as e:
            print(e)
        except Exception as e:
            print(f"Error viewing Adoption table: {e}")

    def insert_adoption(self, petname, petage, petbreed, name):
        try:
            self.open_connection()
            insert_adoption_query = "INSERT INTO Adopt (petname, petage, petbreed,
name) VALUES (%s, %s, %s, %s);"
            self.stmt.execute(insert_adoption_query, (petname, petage, petbreed, name))
            print("Adoption data inserted successfully.")
            self.conn.commit()
            self.close_connection()
        except Exception as e:
            print(f"Error inserting data into Adopt table: {e}")
```

```
def adopt(self):
    try:
        self.pet_shelter_dao.list_available_pets()
        self.get_participants()
        pet_id = int(input("Enter pet ID: "))
        self.open_connection()
        select_query = "SELECT * FROM Pets WHERE id=%s"
        self.stmt.execute(select_query, (pet_id,))
        records = self.stmt.fetchall()
        if not records:
            raise AdoptionException("Pet not found in the shelter.")
        record = records[0]
        petname = record[1]
        petage = record[2]
        petbreed = record[3]
        self.close_connection()

        participant_id = int(input("Enter participant ID: "))
        self.open_connection()
        select_query = "SELECT * FROM Participants WHERE ID=%s"
        self.stmt.execute(select_query, (participant_id,))
        records = self.stmt.fetchall()
        if not records:
            raise AdoptionException("Participant not found.")
        name = records[0][1]
        self.close_connection()

        self.insert_adoption(petname, petage, petbreed, name)
        print(f'{name} adopted {petname} successfully')

        self.open_connection()
        delete_query = f"DELETE FROM Pets WHERE id = {pet_id}"
        self.stmt.execute(delete_query)
        self.conn.commit()
        print("Pet is successfully removed from the shelter!")
        self.close_connection()

    except AdoptionException as e:
        print(e)
    except Exception as e:
        print(f"Error during adoption: {e}")
```

```python
  def get_participants(self):
      try:
          self.open_connection()
          select_query = "SELECT * FROM Participants"
          self.stmt.execute(select_query)
          records = self.stmt.fetchall()
          for record in records:
              print(record)
          self.close_connection()
      except Exception as e:
          print(f"Error getting participants: {e}")
```

```
Adoption table data:
('Buddy', 3, 'Golden Retriever', 'John Doe')
('Cleo', 2, 'Siamese', 'Jane Smith')
('Max', 5, 'Labrador Retriever', 'Michael Johnson')
('Whiskers', 1, 'Persian', 'Emily Davis')
('Rocky', 4, 'Poodle', 'David Wilson')
('Simba', 2, 'Tabby', 'Sophia Thompson')
('Lucy', 6, 'Beagle', 'Daniel Anderson')
('Smokey', 3, 'Russian Blue', 'Olivia Martinez')
('Daisy', 1, 'Chihuahua', 'William Taylor')
('Marley', 4, 'Boxer', 'Emma Brown')
('Lucy', 6, 'Beagle', 'Jane Smith')
('rocky', 2, 'labrador', 'anish')
('adfn', 3, 'sdf', 'anish')
('df', 4, 'sdf', 'anish')
('tommy', 2, 'labrador', 'Emma Brown')
```

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

File Handling Exception: In the Pet Adoption Platform, there might be scenarios where the program needs to read data from a file (e.g., a list of pets in a shelter). Write a program that attempts to read data from a file. Implement exception handling to catch any file-related exceptions (e.g., FileNotFoundException) and display an error message if the file is not found or cannot be read.

```
class FileHandlingException(Exception):
   def __init__(self, message="No data Found in Adoption Table"):
       self.message = message
       super().__init__(self.message)
```

```
class InsufficientFundsException(Exception):
   def __init__(self, message="Insufficient funds for donation (amount should be at
least 100)"):
       self.message = message
       super().__init__(self.message)
```

```
class FileHandlingException(Exception):
   def __init__(self, message="No data Found in Adoption Table"):
       self.message = message
       super().__init__(self.message)
```

```
PetPals: The Pet Adoption Platform
1. Add a pet to the shelter
2. List available pets in the shelter
3. Record a cash donation
4. View donation data
5. Host an adoption event
6. Register for an adoption event
7. View adoption data
8. Adopt a pet
9. Exit
Enter your choice (1-9): 3
Enter donor name: nadhin
Enter donation amount: 99
Error recording cash donation: Insufficient funds for donation (amount should be at least 100)
```

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

```
class AdoptionException(Exception):
   def __init__(self, message="This pet is already adopted"):
       self.message = message
       super().__init__(self.message)
```

```
PetPals: The Pet Adoption Platform
1. Add a pet to the shelter
2. List available pets in the shelter
3. Record a cash donation
4. View donation data
5. Host an adoption event
6. Register for an adoption event
7. View adoption data
8. Adopt a pet
9. Exit
Enter your choice (1-9): 8
--Database is Connected--
Available Pets:
Buddy, 3 years old, Golden Retriever
Cleo, 2 years old, Siamese
Max, 5 years old, Labrador Retriever
Whiskers, 1 years old, Persian
Rocky, 4 years old, Poodle
Simba, 2 years old, Tabby
Smokey, 3 years old, Russian Blue
Daisy, 1 years old, Chihuahua
Marley, 4 years old, Boxer
n, 2 years old, df
a, 2 years old, df
tom, 3 years old, lab
tiger, 4 years old, labrador
--Database is Connected--
(1, 'John Doe')
(2, 'Jane Smith')
(3, 'Michael Johnson')
(4, 'Emily Davis')
(5, 'David Wilson')
(6, 'Sophia Thompson')
(7, 'Daniel Anderson')
(8, 'Olivia Martinez')
(9, 'William Taylor')
(10, 'Emma Brown')
(11, 'anish')
(12, 'anish')
Enter pet ID: 19
--Database is Connected--
Pet not found in the shelter.
```

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.

```python
import mysql.connector
from util.propertyutil import PropertyUtil

class DBConnection:
   def __init__(self):
       self.conn = None
       self.stmt = None


   def open_connection(self):
       try:
           host, username, password, database = PropertyUtil.get_property_string()
           self.conn = mysql.connector.connect(
               host=host,
               user=username,
               password=password,
               database=database
           )
           if self.conn :
               print("--Database is Connected--")

           self.stmt = self.conn.cursor()
       except Exception as e:
           print(e)

   def close_connection(self):
       try:
           self.conn.close()

       except Exception as e:
           print(e)
```

```
Enter your choice (1-9): 2
--Database is Connected--
Available Pets:
Buddy, 3 years old, Golden Retriever
Cleo, 2 years old, Siamese
Max, 5 years old, Labrador Retriever
Whiskers, 1 years old, Persian
Rocky, 4 years old, Poodle
Simba, 2 years old, Tabby
Smokey, 3 years old, Russian Blue
Daisy, 1 years old, Chihuahua
```

Table: pet (not exist)

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

```python
from entity.cashdonation import CashDonation
from exception.insufficientfundserror import InsufficientFundsException
from exception.invalidnameerror import InvalidNameError
from exception.invalidamounterror import InvalidAmountError
import mysql.connector as sql
from util.dbconnutil import DBConnection

class CashDonationDAO(DBConnection):
    def __init__(self):
        self.result_list = []

    def create_table(self):
        try:
            self.open_connection()
            create_table_query = '''
                CREATE TABLE IF NOT EXISTS CashDonation (
                    id INT PRIMARY KEY AUTO_INCREMENT,
                    DonorName VARCHAR(255) NOT NULL,
                    Amount DECIMAL(10, 2) NOT NULL
                );
            '''
```

```
            self.stmt.execute(create_table_query)
            print("CashDonation table is created.")
            self.close_connection()
        except Exception as e:
            print(f"Error creating CashDonation table: {e}")

    def record_donation(self, donor_name, amount):
        try:
            if not isinstance(donor_name, str):
                raise InvalidNameError()
            for char in donor_name:
                if not char.isalpha() and not char.isspace():
                    raise InvalidNameError()
            if not isinstance(amount, (int, float)) or amount <= 0:
                raise InvalidAmountError()
            elif amount < 100:
                raise InsufficientFundsException()

            cash_donation = CashDonation(donor_name, amount)
            self.open_connection()
            insert_query = "INSERT INTO CashDonation (DonorName, Amount) VALUES (%s,
%s)"
            self.stmt.execute(insert_query, (cash_donation.donor_name,
cash_donation.amount))
            self.conn.commit()
            print("Cash donation recorded successfully.")
            self.close_connection()
        except Exception as e:
            print(f"Error recording cash donation: {e}")

    def view_donation_data(self):
        try:
            self.open_connection()
            select_query = "SELECT * FROM CashDonation"
            self.stmt.execute(select_query)
            records = self.stmt.fetchall()
            self.result_list = []
            for record in records:
                self.result_list.append({
                    "id": record[0],
                    "donor_name": record[1],
                    "amount": record[2]
```

```
            })
        self.close_connection()
        return self.result_list
    except Exception as e:
        print(f"Error selecting from CashDonation table: {e}")
```

```
PetPals: The Pet Adoption Platform
1. Add a pet to the shelter
2. List available pets in the shelter
3. Record a cash donation
4. View donation data
5. Host an adoption event
6. Register for an adoption event
7. View adoption data
8. Adopt a pet
9. Exit
Enter your choice (1-9): 3
Enter donor name: kaviyana
Enter donation amount: 2000
--Database is Connected--
Cash donation recorded successfully.
```

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.

```
from dao.interface.iadoptable import IAdoptable
from exception.invalidnameerror import InvalidNameError
from exception.adoptionerror import AdoptionException
from exception.filehandlingerror import FileHandlingException
import mysql.connector as sql
from util.dbconnutil import DBConnection
from dao.petshelter import PetShelterDAO

class AdoptionEventDAO(DBConnection, IAdoptable):
   def __init__(self):
       self.pet_shelter_dao = PetShelterDAO()
```

```
def create_event(self):
    try:
        self.open_connection()
        create_event_query = '''
            CREATE TABLE IF NOT EXISTS Event (
                ID INT PRIMARY KEY AUTO_INCREMENT,
                Details VARCHAR(255) NOT NULL
            );
        '''
        self.stmt.execute(create_event_query)

        # self.close_connection()
    except Exception as e:
        print(f"Error creating Event table: {e}")

def create_participants(self):
    try:
        self.open_connection()
        create_participants_query = '''
            CREATE TABLE IF NOT EXISTS Participants (
                ID INT PRIMARY KEY AUTO_INCREMENT,
                Name VARCHAR(255) NOT NULL
            );
        '''
        self.stmt.execute(create_participants_query)

        # self.close_connection()
    except Exception as e:
        print(f"Error creating Participants table: {e}")

def create_adoption(self):
    try:
        self.open_connection()
        create_adoption_query = '''
            CREATE TABLE IF NOT EXISTS Adopt (
                petname VARCHAR(50),
                petage INTEGER,
                petbreed VARCHAR(50),
                name VARCHAR(50)
            );
        '''
        self.stmt.execute(create_adoption_query)
```

```
            # self.close_connection()
        except Exception as e:
            print(f"Error creating Adopt table: {e}")

    def register_participant(self):
        try:
            participant_name = input("Enter participant name: ")
            if not isinstance(participant_name, str):
                raise InvalidNameError()
            for char in participant_name:
                if not char.isalpha() and not char.isspace():
                    raise InvalidNameError()
            self.open_connection()
            insert_query = "INSERT INTO Participants (Name) VALUES (%s)"
            self.stmt.execute(insert_query, (participant_name,))
            self.conn.commit()
            print(f"Participant '{participant_name}' added successfully.")
            # self.close_connection()
        except Exception as e:
            print(f"Error adding participant: {e}")

    def host_event(self):
        try:
            event_details = input("Enter event details: ")
            self.open_connection()
            insert_query = "INSERT INTO Event (Details) VALUES (%s)"
            self.stmt.execute(insert_query, (event_details,))
            self.conn.commit()
            print("Event hosted successfully.")
            # self.close_connection()
        except Exception as e:
            print(f"Error hosting event: {e}")

    def view_adoption(self):
        try:
            self.open_connection()
            view_adoption_query = "SELECT * FROM Adopt;"
            self.stmt.execute(view_adoption_query)
            result = self.stmt.fetchall()
            if result:
                print("Adoption table data:")
```

```
                for row in result:
                    print(row)
            else:
                raise FileHandlingException()
            # self.close_connection()
        except FileHandlingException as e:
            print(e)
        except Exception as e:
            print(f"Error viewing Adoption table: {e}")

    def insert_adoption(self, petname, petage, petbreed, name):
        try:
            self.open_connection()
            insert_adoption_query = "INSERT INTO Adopt (petname, petage, petbreed,
name) VALUES (%s, %s, %s, %s);"
            self.stmt.execute(insert_adoption_query, (petname, petage, petbreed, name))
            print("Adoption data inserted successfully.")
            self.conn.commit()
            self.close_connection()
        except Exception as e:
            print(f"Error inserting data into Adopt table: {e}")

    def adopt(self):
        try:
            self.pet_shelter_dao.list_available_pets()
            self.get_participants()
            pet_id = int(input("Enter pet ID: "))
            self.open_connection()
            select_query = "SELECT * FROM Pets WHERE id=%s"
            self.stmt.execute(select_query, (pet_id,))
            records = self.stmt.fetchall()
            if not records:
                raise AdoptionException("Pet not found in the shelter.")
            record = records[0]
            petname = record[1]
            petage = record[2]
            petbreed = record[3]
            self.close_connection()

            participant_id = int(input("Enter participant ID: "))
            self.open_connection()
            select_query = "SELECT * FROM Participants WHERE ID=%s"
```

```
            self.stmt.execute(select_query, (participant_id,))
            records = self.stmt.fetchall()
            if not records:
                raise AdoptionException("Participant not found.")
            name = records[0][1]
            self.close_connection()

            self.insert_adoption(petname, petage, petbreed, name)
            print(f'{name} adopted {petname} successfully')

            self.open_connection()
            delete_query = f"DELETE FROM Pets WHERE id = {pet_id}"
            self.stmt.execute(delete_query)
            self.conn.commit()
            print("Pet is successfully removed from the shelter!")
            self.close_connection()

        except AdoptionException as e:
            print(e)
        except Exception as e:
            print(f"Error during adoption: {e}")

    def get_participants(self):
        try:
            self.open_connection()
            select_query = "SELECT * FROM Participants"
            self.stmt.execute(select_query)
            records = self.stmt.fetchall()
            for record in records:
                print(record)
            self.close_connection()
        except Exception as e:
            print(f"Error getting participants: {e}")
```

```
Enter pet ID: 11
--Database is Connected--
Enter participant ID: 11
--Database is Connected--
--Database is Connected--
Adoption data inserted successfully.
anish adopted n successfully
--Database is Connected--
Pet is successfully removed from the shelter!

PetPals: The Pet Adoption Platform
1. Add a pet to the shelter
2. List available pets in the shelter
3. Record a cash donation
4. View donation data
5. Host an adoption event
6. Register for an adoption event
7. View adoption data
8. Adopt a pet
9. Exit
Enter your choice (1-9):
```