CODING CHALLENGE

PetPals

Name : K.MEGHASYAM

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:
    def __init__(self, name, age, breed):

        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

    def set_age(self, age):
        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



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


pet1 = Pet("snoopy", 3, 'Dog')
pet2 = Pet("Browny", 5, 'Dog')
print(pet1.to_string())
print(pet2.to_string())
```

```
snoopy, 3 years old, Dog
Browny, 5 years old, Dog
```

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.

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

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


dog1 = Dog("snoopy", 3, "Dog", "Bull Dog")
dog2 = Dog("Browny", 5, "Dog", "Golden")
print(dog1.to_string())
print(dog2.to_string())
```

```
snoopy, 3 years old, belongs to Dog, Bull Dog.
Browny, 5 years old, belongs to Dog, Golden.
```

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

cat1 = Cat("misty", 3, "Cat", "orange")
cat2 = Cat("sammy", 1, "Cat", "brown")
print(cat1.to_string())
print(cat2.to_string())
```

```
misty, 3 years old, belongs to Cat, is of orange in color
sammy, 1 years old, belongs to Cat, is of brown in color
```

3.PetShelter Class:

Attributes:

• availablePets (List of Pet): A list to store available pets for adoption.

Methods:

• AddPet(Pet pet): Adds a pet to the list of available pets.

• RemovePet(Pet pet): Removes a pet from the list of available pets.

• ListAvailablePets(): Lists all available pets in the shelter.

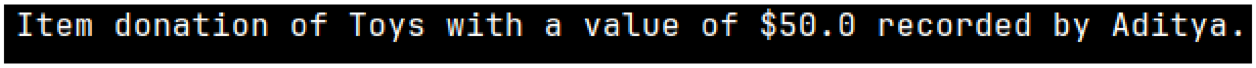
```
from pet import Pet
class PetShelter():
    def __init__(self):
        self.available_pets = []

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

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

    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)
obj=PetShelter()
```

```
dog1 = Dog("snoopy", 3, "Dog", "Bull Dog")
cat1 = Cat("misty", 3, "Cat", "orange")
obj.add_pet(dog1)
obj.add_pet(cat1)
obj.list_available_pets()
obj.remove_pet(dog1)
print("After dog Removal:")
obj.list_available_pets()
```

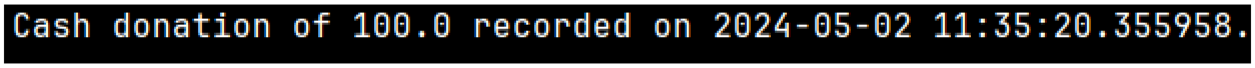
```
Available Pets:
 snoopy, 3 years old, Dog
 misty, 3 years old, Cat
After dog Removal:
 misty, 3 years old, Cat
```

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

```
from abc import ABC, abstractmethod
from entity.donation import Donation
from datetime import datetime

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

     @abstractmethod
    def RecordDonation(self):
        pass

class CashDonation(Donation):
    def __init__(self, donor_name, amount, donation_date):
        super().__init__(donor_name, amount)
        self.donation_date = donation_date
    def record_donation(self):
        print(f"Cash donation of {self.amount} recorded
on{self.donation_date}.")


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):
        # Implement item donation recording logic here
        print(f"Item donation of {self.item_type} recorded.")


item_donation = ItemDonation("Aditya", 50.0, "Toys")
item_donation.record_donation()

cash_donation = CashDonation("Aditya", 100.0, datetime.now())
cash_donation.record_donation()
```



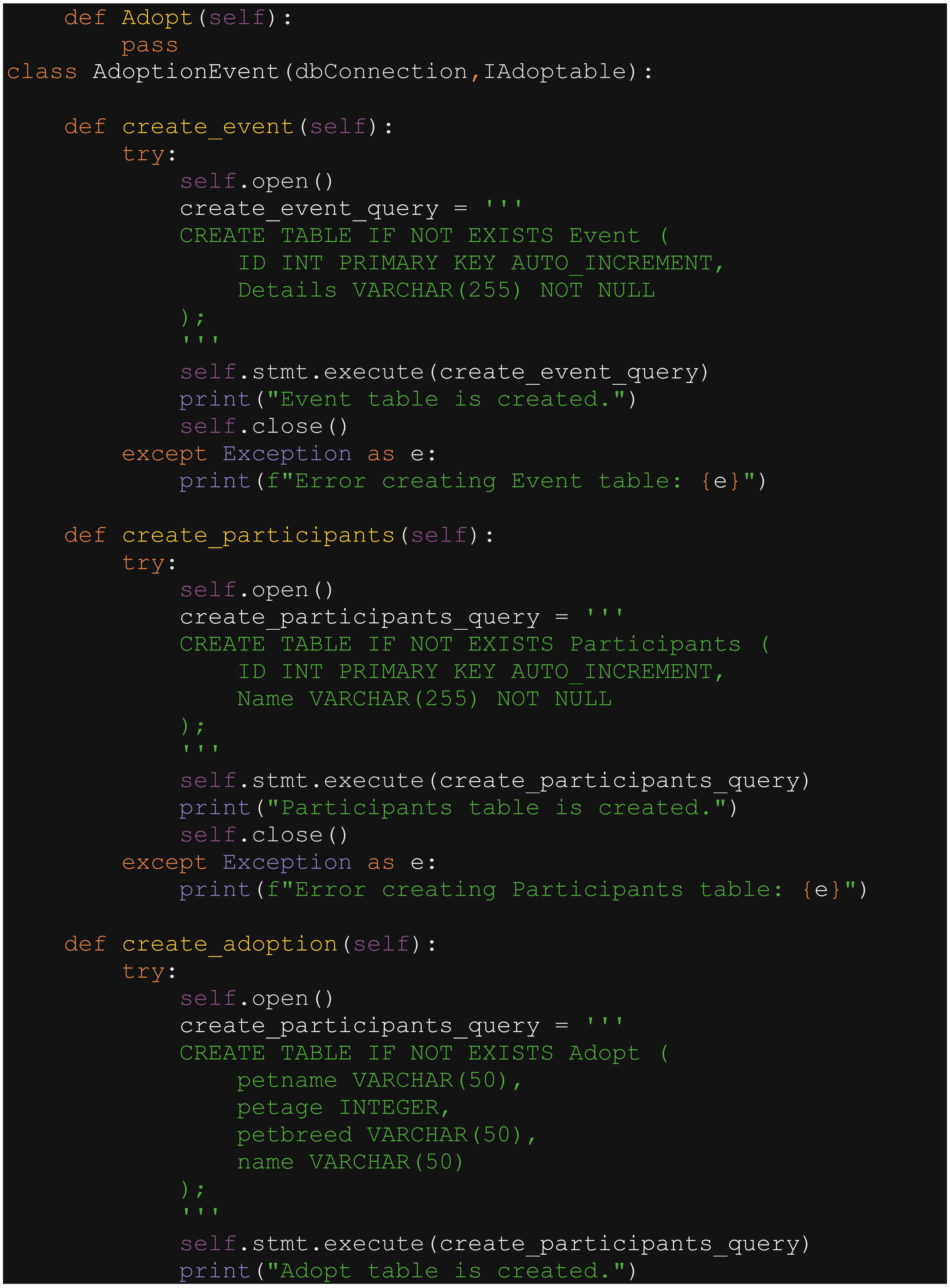
```
Item donation of Toys with a value of $50.0 recorded by Aditya.
```



```
Cash donation of 100.0 recorded on 2024-05-02 11:35:20.355958.
```

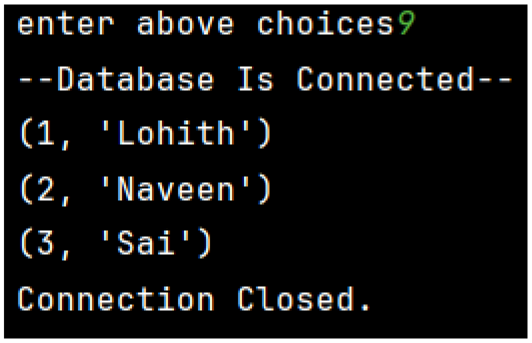
5.IAdoptable Interface/Abstract Class: Methods: • Adopt(): An abstract method to handle the adoption process. AdoptionEvent Class: Attributes: • Participants (List of IAdoptable): A list of participants (shelters and adopters) in the adoption event. Methods: • HostEvent(): Hosts the adoption event. • RegisterParticipant(IAdoptable participant): Registers a participant for the event

```
from abc import ABC, abstractmethod
import mysql.connector as sql
from entity.IAdoptable import IAdoptable
from exception.InvalidNameError import InvalidNameError
from util.DBConnUtil import dbConnection
from dao.petshelter import *

class IAdoptable(ABC):
    @abstractmethod
```

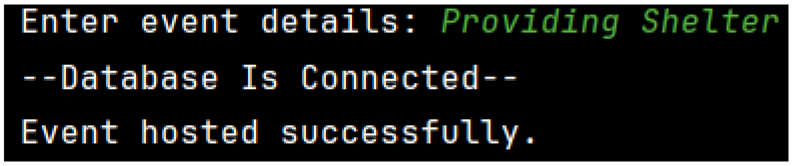


```python
    def Adopt(self):
        pass
class AdoptionEvent(dbConnection,IAdoptable):

    def create_event(self):
        try:
            self.open()
            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)
            print("Event table is created.")
            self.close()
        except Exception as e:
            print(f"Error creating Event table: {e}")

    def create_participants(self):
        try:
            self.open()
            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)
            print("Participants table is created.")
            self.close()
        except Exception as e:
            print(f"Error creating Participants table: {e}")

    def create_adoption(self):
        try:
            self.open()
            create_participants_query = '''
            CREATE TABLE IF NOT EXISTS Adopt (
                petname VARCHAR(50),
                petage INTEGER,
                petbreed VARCHAR(50),
                name VARCHAR(50)
            );
            '''
            self.stmt.execute(create_participants_query)
            print("Adopt table is created.")
            self.close()
        except Exception as e:
            print(f"Error creating Participants table: {e}")

    def RegisterParticipant(self):
        try:
            participant_name = input("Enter participant name: ")

            if not isinstance(participant_name, str):
                raise InvalidNameError()
            for i in participant_name:
                if not i.isalpha() and not i.isspace():
                    raise InvalidNameError()

            self.open()
            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()
        except Exception as e:
            print(f"Error adding participant: {e}")

    def HostEvent(self):
        try:
            event_details = input("Enter event details: ")

            # Add the event to the database
            self.open()
            insert_query = "INSERT INTO Event (Details) VALUES
(%s)"
            self.stmt.execute(insert_query, (event_details,))
            self.conn.commit()
            print("Event hosted successfully.")
            self.close()
        except Exception as e:
            print(f"Error hosting event: {e}")


    def ViewAdoption(self):
        try:
            self.open()
            view_adoption_query = "SELECT * FROM Adopt;"
            self.stmt.execute(view_adoption_query)
            result = self.stmt.fetchall()
```

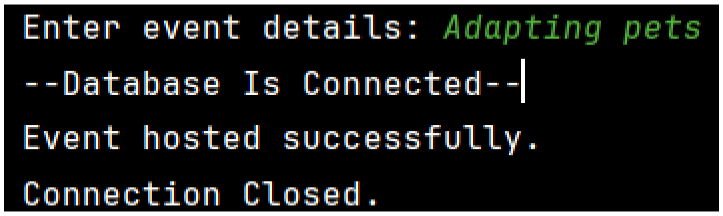
```python
            if result:
                print("Adoption table data:")
                for row in result:
                    print(row)
            else:
                print("No data found in Adoption table.")

            self.close()
        except Exception as e:
            print(f"Error viewing Adoption table: {e}")

    def InsertAdoption(self, petname, petage, petbreed, name):
        try:
            self.open()
            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()
        except Exception as e:
            print(f"Error inserting data into Adopt table: {e}")

    def Adopt(self):
        try:
            self.open()
            select_query = "SELECT * FROM Pets"
            self.stmt.execute(select_query)
            records = self.stmt.fetchall()
            for i in records:
                print(i)
        except sql.Error as e:
            print(f"Error listing available pets: {e}")
        self.GetParticipants()

        try:
            id = int(input("enter petID"))
            self.open()
            select_query = "SELECT * FROM Pets where id=%s"
            self.stmt.execute(select_query,(id,))
            records = self.stmt.fetchall()[0]
            print(records)
            petID = records[0]
            petname = records[1]
            petage = records[2]
            petbreeed = records[3]
            self.close()
            nameid = int(input("enter participantID"))
            self.open()
            select_query = "SELECT * FROM Participants where
ID=%s"
            self.stmt.execute(select_query,(nameid,))
            records = self.stmt.fetchall()[0]
            name = records[1]
            print(name)
            self.close()
            self.InsertAdoption(petname, petage, petbreeed,
name)
            print(f'{name} adopted {petname} successfully')
            delete_query = f"DELETE FROM Pets WHERE id =
{petID}"
            self.open()
            self.stmt.execute(delete_query)
            self.conn.commit()
            print("Pet is successfully removed from Shelter!!")

        except Exception as e:
            print(f"Error getting participants: {e}")
```



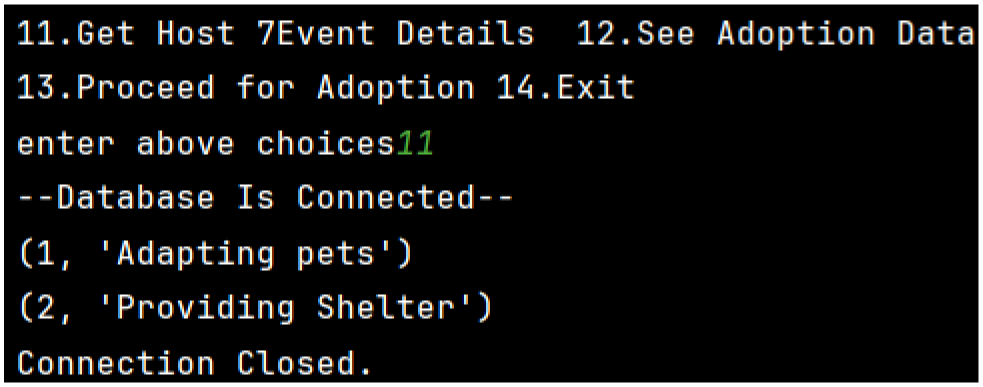
```
enter above choices9
--Database Is Connected--
(1, 'Lohith')
(2, 'Naveen')
(3, 'Sai')
Connection Closed.
```



```
Enter event details: Providing Shelter
--Database Is Connected--
Event hosted successfully.
```



```
Enter event details: Adapting pets
--Database Is Connected--
Event hosted successfully.
Connection Closed.
```



```
11.Get Host 7Event Details  12.See Adoption Data
13.Proceed for Adoption 14.Exit
enter above choices11
--Database Is Connected--
(1, 'Adapting pets')
(2, 'Providing Shelter')
Connection Closed.
```

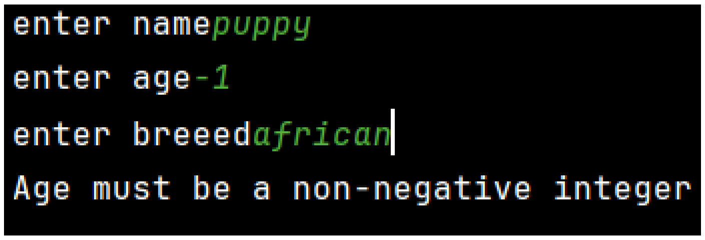
6.Exceptions handling

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

```
class InvalidAgeError(Exception):
    def __init__(self, message="Invalid age for a dog"):
        self.message = message
        super().__init__(self.message)

from exception.InvalidAgeError import *


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.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

    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

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

try:

    pet1 = Pet("snoopy", 3, 'Dog')

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

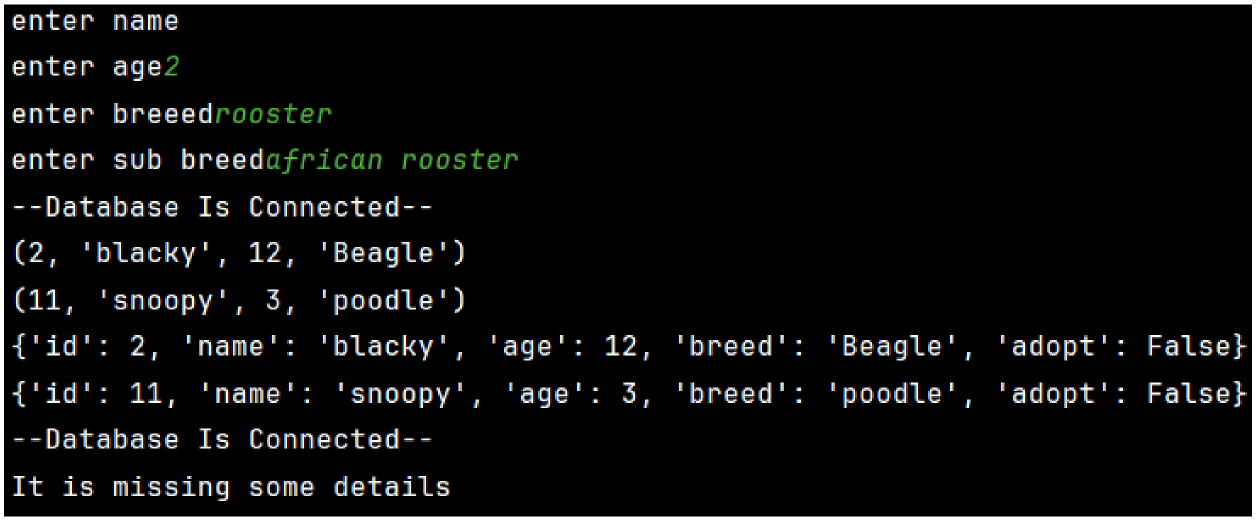


```
enter namepuppy
enter age-1
enter breeedafrican
Age must be a non-negative integer
```

• **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.

```
from entity.pet import Pet
import mysql.connector as sql
from exception.NullReferenceException import *
from util.DBConnUtil import dbConnection

class NullReferenceException(Exception):
    def __init__(self, message="It is missing some details"):
        self.message = message
        super().__init__(self.message)
def AddPet(self,pet):
    try:
        if not isinstance(pet, Pet):
            raise ValueError("Invalid pet type.")

        # Check if a pet with the same details already exists
        self.ListAvailablePets()
        for existing_pet in self.available_pets:
            print(existing_pet)
            if (
                    existing_pet["name"] == pet.name
                    and existing_pet["age"] == pet.age
                    and existing_pet["breed"] == pet.breed
            ):
                raise DuplicateObjError()

        insert_query = "INSERT INTO Pets (Name, Age, Breed) VALUES (%s, %s,
%s)"
        self.open()
        self.stmt.execute(insert_query, (pet.get_name(), pet.get_age(),
pet.get_breed()))
        if (
            pet.get_name()==" "
            or pet.get_name()==" "
            or pet.get_name()==" "
        ):
            raise NullReferenceException()
        self.conn.commit()
        self.close()

        # Update available_pets list after a successful insert
        self.ListAvailablePets()
```



```
enter name
enter age2
enter breeedrooster
enter sub breedafrican rooster
--Database Is Connected--
(2, 'blacky', 12, 'Beagle')
(11, 'snoopy', 3, 'poodle')
{'id': 2, 'name': 'blacky', 'age': 12, 'breed': 'Beagle', 'adopt': False}
{'id': 11, 'name': 'snoopy', 'age': 3, 'breed': 'poodle', 'adopt': False}
--Database Is Connected--
It is missing some details
```

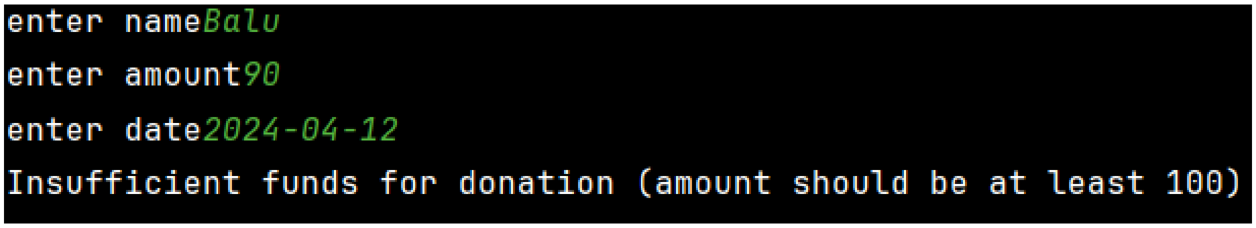
• Insufficient Funds Exception:

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.

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

```python
from entity.donation import Donation
import mysql.connector as sql
from exception.InsufficientFundsException import
InsufficientFundsException
from util.DBConnUtil import dbConnection



class CashDonation(Donation, dbConnection):

    def __init__(self, donor_name=None, amount=None,
donation_date=None):
        if donor_name!=None and amount!=None and
```

```
donation_date!=None:

            if amount < 100:
                raise InsufficientFundsException()
            self.donor_name = donor_name
            self.amount = amount
            self.donation_date = donation_date
            self.result_list = []

    def createTable(self):
        try:
            self.open()
            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()
        except Exception as e:
            print(f"Error creating CashDonation table: {e}")

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

    def ViewAmountDonationData(self):
        try:
            self.open()
            select_query = "SELECT * FROM CashDonation"
            self.stmt.execute(select_query)
            records = self.stmt.fetchall()
            self.result_list = []
            for record in records:
                print(record)
                self.result_list.append({
                    "id": record[0],
                    "donor_name": record[1],
                    "amount": record[2]
                })
            self.close()
            return self.result_list
        except Exception as e:
            print(f"Error selecting from CashDonation table:
{e}")
```

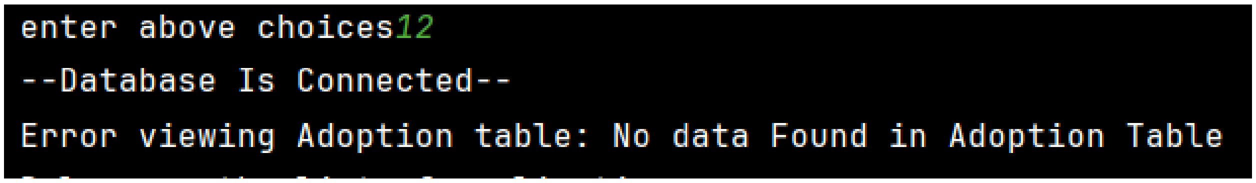


```
enter nameBalu
enter amount90
enter date2024-04-12
Insufficient funds for donation (amount should be at least 100)
```

• 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.

```
import mysql.connector as sql
from exception.AdoptionException import AdoptionException
from entity.IAdoptable import IAdoptable
from exception.FileHandlingException import
FileHandlingException
from util.DBConnUtil import dbConnection

class FileHandlingException(Exception):
    def __init__(self, message="No data Found in Adoption
Table”):
        self.message = message
        super().__init__(self.message)
def ViewAdoption(self):
    try:
        self.open()
        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:
            print("No data found in Adoption table.")

        self.close()
    except Exception as e:
        print(f"Error viewing Adoption table: {e}")
```

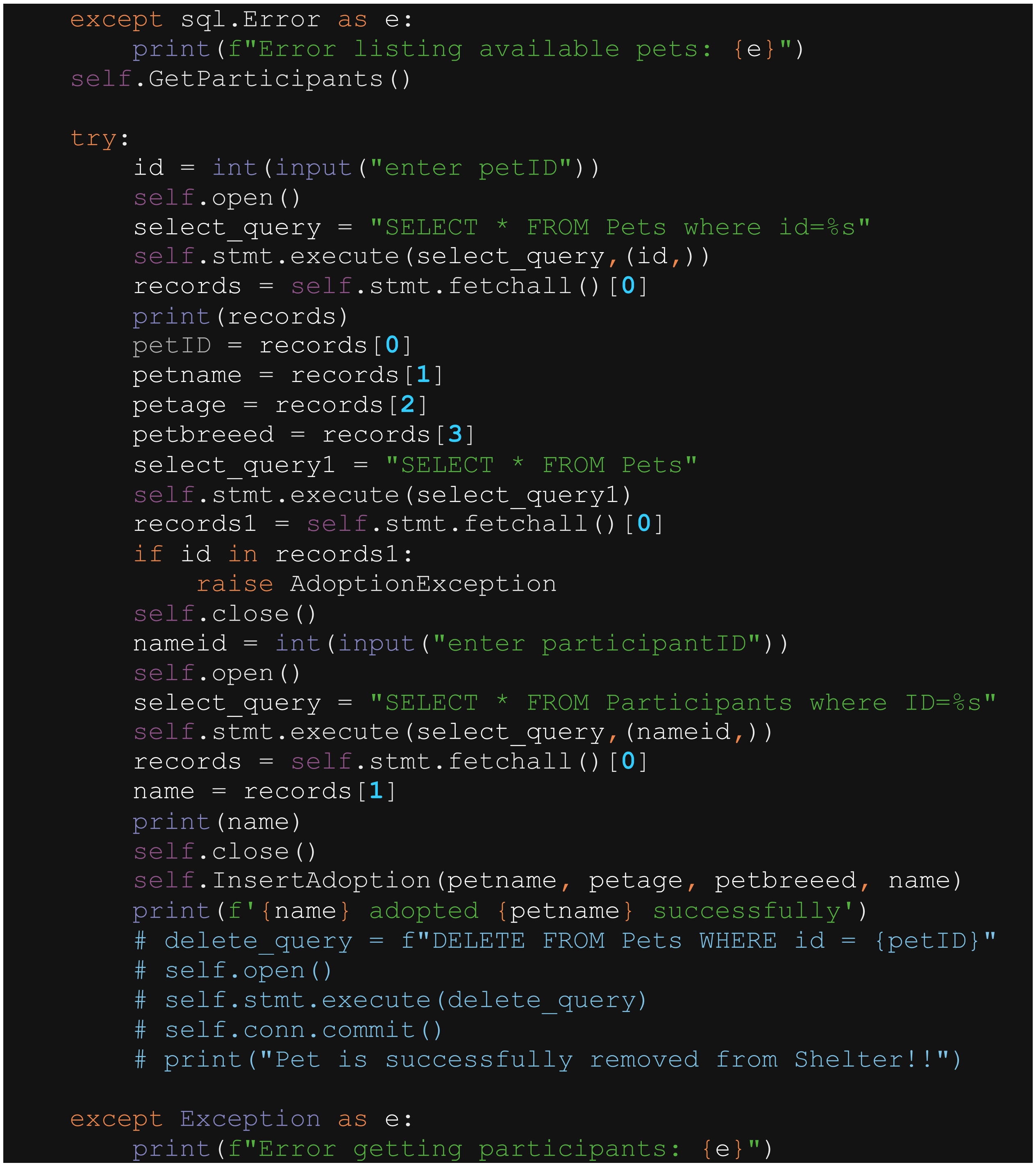


```
enter above choices12
--Database Is Connected--
Error viewing Adoption table: No data Found in Adoption Table
```

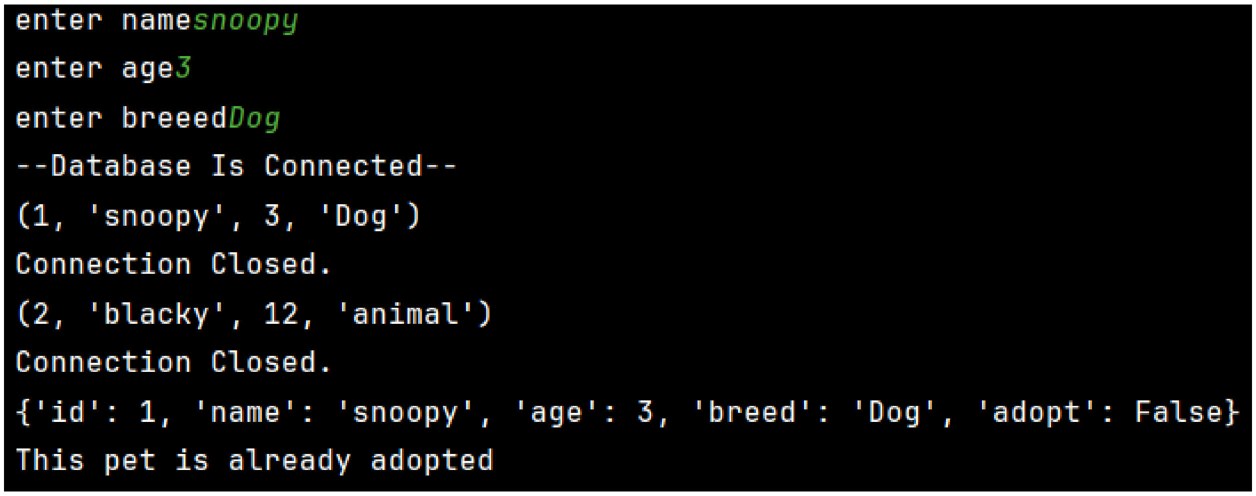
• Custom Exception for Adoption Errors:

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.

```
import mysql.connector as sql
from exception.AdoptionException import AdoptionException
from util.DBConnUtil import dbConnection
from dao.petshelter import *

class AdoptionException(Exception):
    def __init__(self, message="This pet is already adopted"):
        self.message = message
        super().__init__(self.message)
def Adopt(self):
    try:
        self.open()
        select_query = "SELECT * FROM Pets"
        self.stmt.execute(select_query)
        records1 = self.stmt.fetchall()
        for i in records1:
            print(i)
```



```
    except sql.Error as e:
        print(f"Error listing available pets: {e}")
    self.GetParticipants()

    try:
        id = int(input("enter petID"))
        self.open()
        select_query = "SELECT * FROM Pets where id=%s"
        self.stmt.execute(select_query,(id,))
        records = self.stmt.fetchall()[0]
        print(records)
        petID = records[0]
        petname = records[1]
        petage = records[2]
        petbreeed = records[3]
        select_query1 = "SELECT * FROM Pets"
        self.stmt.execute(select_query1)
        records1 = self.stmt.fetchall()[0]
        if id in records1:
            raise AdoptionException
        self.close()
        nameid = int(input("enter participantID"))
        self.open()
        select_query = "SELECT * FROM Participants where ID=%s"
        self.stmt.execute(select_query,(nameid,))
        records = self.stmt.fetchall()[0]
        name = records[1]
        print(name)
        self.close()
        self.InsertAdoption(petname, petage, petbreeed, name)
        print(f'{name} adopted {petname} successfully')
        # delete_query = f"DELETE FROM Pets WHERE id = {petID}"
        # self.open()
        # self.stmt.execute(delete_query)
        # self.conn.commit()
        # print("Pet is successfully removed from Shelter!!")

    except Exception as e:
        print(f"Error getting participants: {e}")
```



```
enter namesnoopy
enter age3
enter breeedDog
--Database Is Connected--
(1, 'snoopy', 3, 'Dog')
Connection Closed.
(2, 'blacky', 12, 'animal')
Connection Closed.
{'id': 1, 'name': 'snoopy', 'age': 3, 'breed': 'Dog', 'adopt': False}
This pet is already adopted
```

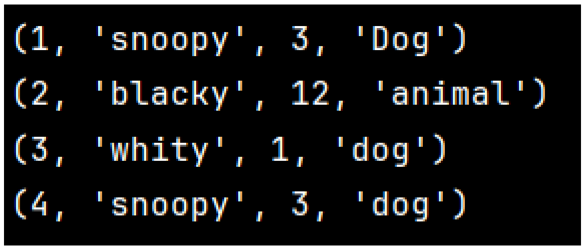
7.Database Connectivity Create and implement the following tasks in your application.

• 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.

```
import mysql.connector as connection

from util.PropertyUtil import PropertyUtil

class dbConnection():
    def _init_(self):
        pass

    def open(self):
        try:
            l = PropertyUtil.getPropertyString()
            self.conn = connection.connect(host=l[0],
database=l[3], username=l[1], password=l[2])
            if self.conn:
                print("--Database Is Connected--")
            self.stmt = self.conn.cursor()
        except Exception as e:
            print(e)

    def close(self):
        try:
            self.conn.close()
            print('Connection Closed.')
        except Exception as e:
            print(e)
```

```
class PropertyUtil:
    def getPropertyString():
        host = 'localhost'
        username = 'root'
        password = 'root'
        database = 'Petpals'
        return host, username, password, database
```

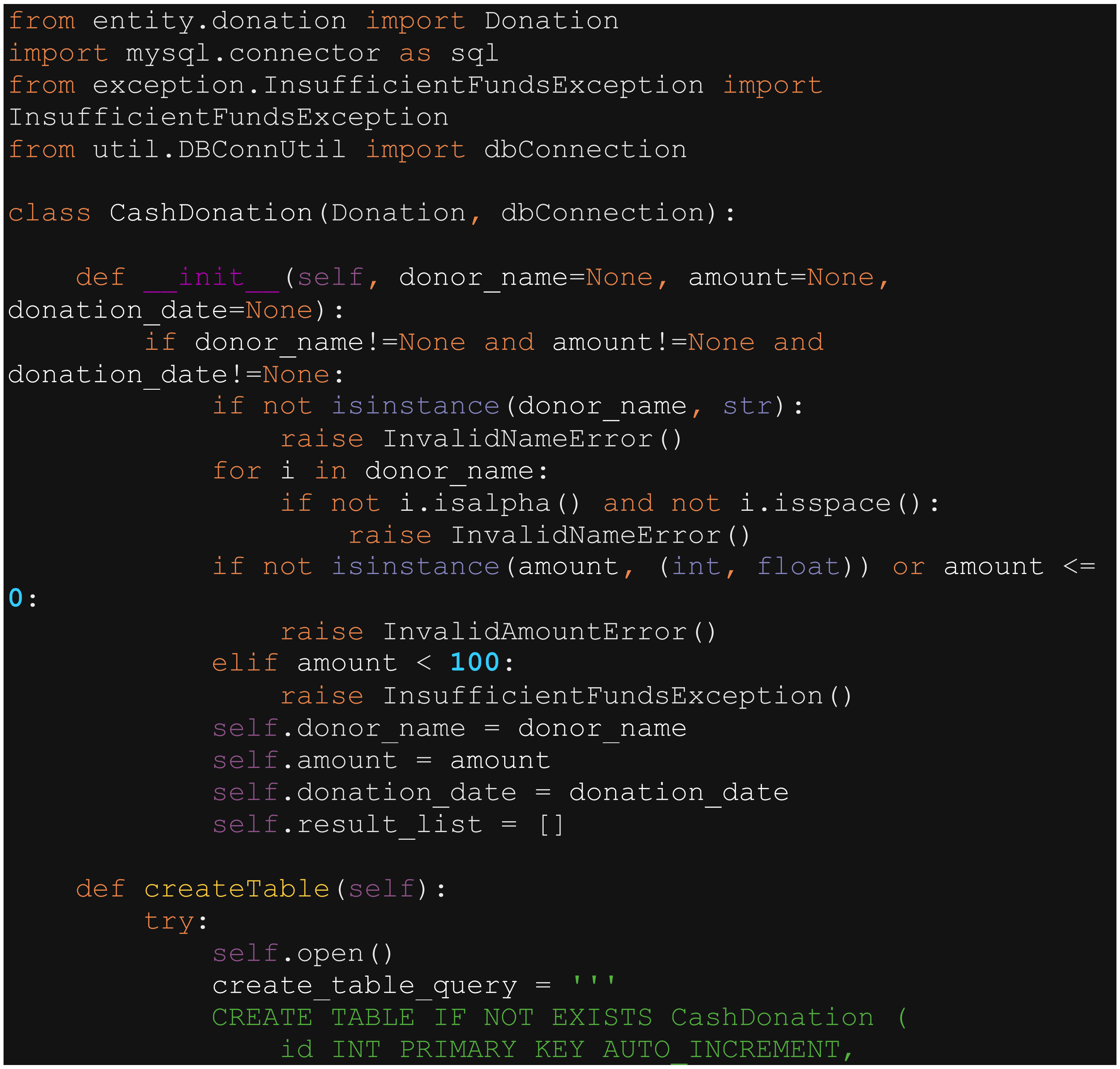
```
from entity.pet import Pet
import mysql.connector as sql
from util.DBConnUtil import dbConnection

class PetShelter(dbConnection):
    available_pets = []

def ListAvailablePets(self):
    try:
        self.open()
        select_query = "SELECT * FROM Pets"
        self.stmt.execute(select_query)
        records = self.stmt.fetchall()
        for i in records:
            self.available_pets.append({
                "id": i[0],
                "name": i[1],
                "age": i[2],
                "breed": i[3],
                "adopt":False
            })
            print(i)
            self.close()
    except sql.Error as e:
        print(f"Error listing available pets: {e}")
```



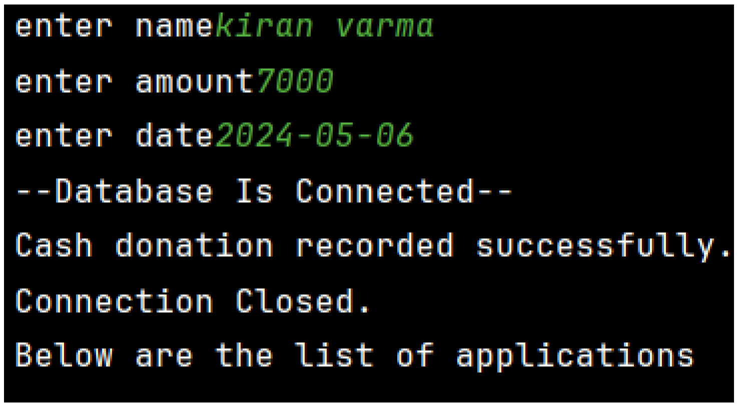
```
(1, 'snoopy', 3, 'Dog')
(2, 'blacky', 12, 'animal')
(3, 'whity', 1, 'dog')
(4, 'snoopy', 3, 'dog')
```

• Donation Recording:

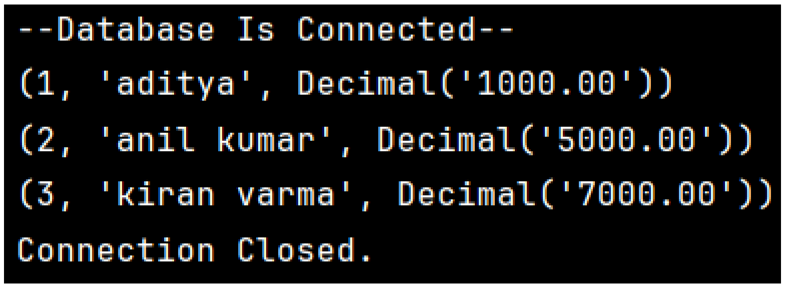
o Create a program that records cash donations made by donors. Allow the user to input donor information and the donation amount and insert this data into the "donations" table in the database. Handle exceptions related to database operations, such as database errors or invalid inputs.



```
from entity.donation import Donation
import mysql.connector as sql
from exception.InsufficientFundsException import
InsufficientFundsException
from util.DBConnUtil import dbConnection

class CashDonation(Donation, dbConnection):

    def __init__(self, donor_name=None, amount=None,
donation_date=None):
        if donor_name!=None and amount!=None and
donation_date!=None:
            if not isinstance(donor_name, str):
                raise InvalidNameError()
            for i in donor_name:
                if not i.isalpha() and not i.isspace():
                    raise InvalidNameError()
            if not isinstance(amount, (int, float)) or amount <=
0:
                raise InvalidAmountError()
            elif amount < 100:
                raise InsufficientFundsException()
            self.donor_name = donor_name
            self.amount = amount
            self.donation_date = donation_date
            self.result_list = []

    def createTable(self):
        try:
            self.open()
            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()
        except Exception as e:
            print(f"Error creating CashDonation table: {e}")

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

    def ViewAmountDonationData(self):
        try:
            self.open()
            select_query = "SELECT * FROM CashDonation"
            self.stmt.execute(select_query)
            records = self.stmt.fetchall()
            self.result_list = []
            for record in records:
                print(record)
                self.result_list.append({
                    "id": record[0],
                    "donor_name": record[1],
                    "amount": record[2]
                })
            self.close()
            return self.result_list
        except Exception as e:
            print(f"Error selecting from CashDonation table: {e}")
```



```
enter namekiran varma
enter amount7000
enter date2024-05-06
--Database Is Connected--
Cash donation recorded successfully.
Connection Closed.
Below are the list of applications
```

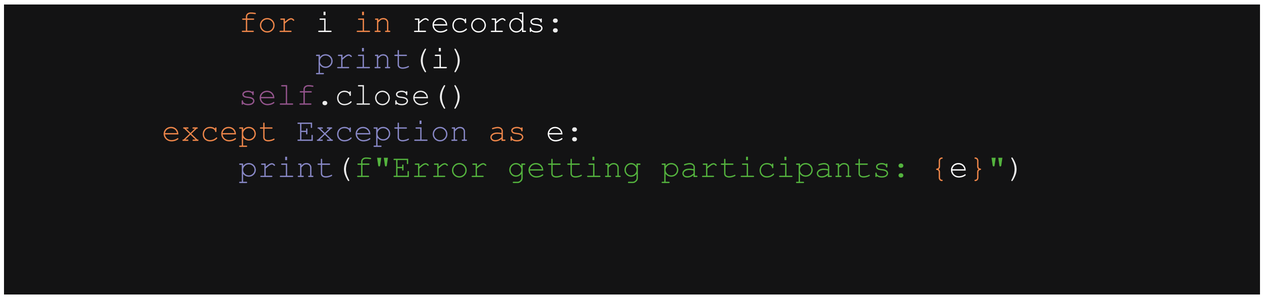


```
--Database Is Connected--
(1, 'aditya', Decimal('1000.00'))
(2, 'anil kumar', Decimal('5000.00'))
(3, 'kiran varma', Decimal('7000.00'))
Connection Closed.
```

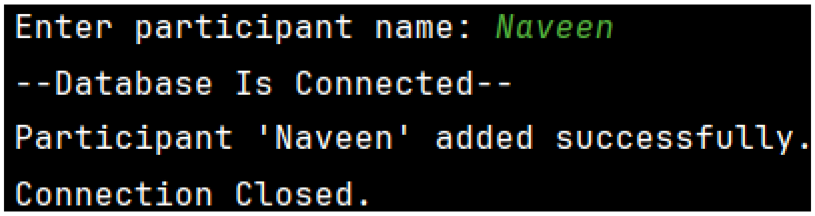
• Adoption Event Management:

o Build a program that connects to the database and retrieves information about upcoming adoption events from the "adoption_events" table. Allow the user to register for an event by adding their details to the "participants" table. Ensure that the program handles database connectivity and insertion exceptions properly.

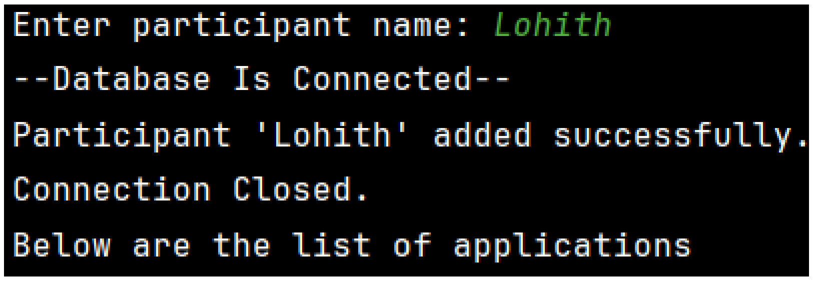
```
import mysql.connector as sql
from entity.IAdoptable import Iadoptablefrom util.DBConnUtil
import dbConnection
from dao.petshelter import *
class AdoptionEvent(dbConnection,IAdoptable):

    def create_participants(self):
        try:
            self.open()
            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)
            print("Participants table is created.")
            self.close()
        except Exception as e:
            print(f"Error creating Participants table: {e}")

    def create_adoption(self):
        try:
            self.open()
            create_participants_query = '''
            CREATE TABLE IF NOT EXISTS Adopt (
                petname VARCHAR(50),
                petage INTEGER,
                petbreed VARCHAR(50),
                name VARCHAR(50)
            );
            '''
            self.stmt.execute(create_participants_query)
            print("Adopt table is created.")
            self.close()
        except Exception as e:
            print(f"Error creating Participants table: {e}")

    def RegisterParticipant(self):
        try:
            participant_name = input("Enter participant name: ")

            if not isinstance(participant_name, str):
                raise InvalidNameError()
            for i in participant_name:
                if not i.isalpha() and not i.isspace():
                    raise InvalidNameError()

            self.open()
            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()
        except Exception as e:
            print(f"Error adding participant: {e}")

    def GetParticipants(self):
        try:
            self.open()
            select_query = "SELECT * FROM Participants"
            self.stmt.execute(select_query)
            records = self.stmt.fetchall()
```



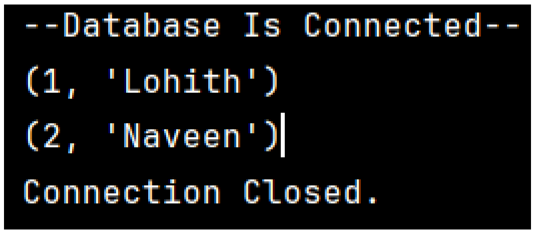
```
            for i in records:
                print(i)
            self.close()
        except Exception as e:
            print(f"Error getting participants: {e}")
```



```
Enter participant name: Naveen
--Database Is Connected--
Participant 'Naveen' added successfully.
Connection Closed.
```



```
Enter participant name: Lohith
--Database Is Connected--
Participant 'Lohith' added successfully.
Connection Closed.
Below are the list of applications
```



```
--Database Is Connected--
(1, 'Lohith')
(2, 'Naveen')
Connection Closed.
```