PETPALS CODING CHALLENGE

(--BY NEEHARIKA MORLA--)

dao:

adoptionevent.py

```
import mysql.connector as sql

from entity.IAdoptable import IAdoptable
from exception.exception import *
from util.dbConnection import dbConnection
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 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}")

    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 GetEvent(self):
        try:
            self.open()
            select_query = "SELECT * FROM Event"
            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 events: {e}")

    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}")

    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}")
```

cashdonation.py

```
from entity.donation import Donation
import mysql.connector as sql
from exception.exception import *
from util.dbConnection 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}")
```

itemdonation.py

```
from entity.donation import Donation
from exception.exception import *
from util.dbConnection import dbConnection
class ItemDonation(Donation,dbConnection):
    donation_data = []

    def __init__(self, donor_name=None, amount=None, itemtype=None):
        if donor_name != None and amount != None and itemtype != 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 <= 100:
                raise InvalidAmountError()
            self.donor_name = donor_name
            self.amount = amount
            self.itemtype = itemtype

    def createTable(self):
        try:
            self.open()
            create_table_query = '''
            CREATE TABLE IF NOT EXISTS ItemDonation (
                id INT PRIMARY KEY AUTO_INCREMENT,
                donor_name VARCHAR(255) NOT NULL,
                amount DECIMAL(10, 2) NOT NULL,
                itemtype VARCHAR(50) NOT NULL
            );
            '''
            self.stmt.execute(create_table_query)
            print("ItemDonation table is created.")
            self.close()
        except Exception as e:
            print(f"Error creating ItemDonation table: {e}")

    def ViewItemDonationData(self):
        try:
            self.open()
            select_query = "SELECT * FROM ItemDonation;"
            self.stmt.execute(select_query)
            result = self.stmt.fetchall()

            if result:
                print("ItemDonation table data:")
                for row in result:
                    print(f"ID: {row[0]}, Donor Name: {row[1]}, Amount: {row[2]},
ItemType: {row[3]}")
            else:
                print("ItemDonation table is empty.")

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

    def RecordDonation(self):
        try:
            self.open()
            insert_query = '''
            INSERT INTO ItemDonation (donor_name, amount, itemtype)
            VALUES (%s, %s, %s);
            '''
            values = (self.donor_name, self.amount, self.itemtype)
            self.stmt.execute(insert_query, values)
            self.conn.commit()
```

```
            print("Record added to ItemDonation table.")
            self.close()
        except Exception as e:
            print(f"Error adding record to ItemDonation table: {e}")
```

petshelter.py

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

class PetShelter(dbConnection):
    available_pets = []

    def create_table(self):
        create_table_query = '''
        CREATE TABLE IF NOT EXISTS Pets (
            id INT PRIMARY KEY AUTO_INCREMENT,
            Name VARCHAR(255) NOT NULL,
            Age INT,
            Breed VARCHAR(255) NOT NULL
        );
        '''
        self.open()
        self.stmt.execute(create_table_query)
        print("Table Pets is created.")
        self.close()

    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()))
            self.conn.commit()
            self.close()

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

            print(f"{pet.get_name()} added to the list of available pets.")
        except sql.Error as e:
            print(f"Error adding pet: {e}")
        except DuplicateObjError as e:
            print("Duplicate pet object. Pet not added.")
        except ValueError as ve:
            print(f"Error: {ve}")

    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}")

    def get_pet_id(self, pet):
        self.ListAvailablePets()
        for record in self.available_pets:
            if record["name"] == pet.name and record["age"] == pet.age and
record["breed"] == pet.breed:
                return record["id"]
        return None

    def RemovePet(self, petid):
        try:
            if isinstance(petid, int) or petid<0:
                if petid:
                    delete_query = f"DELETE FROM Pets WHERE id = {petid}"
                    self.open()
                    self.stmt.execute(delete_query)
                    self.conn.commit()
                    self.close()
                    self.ListAvailablePets()
                    print("deleted successfully!!")
                else:
                    print(f"not found in the list.")
            else:
                print("Invalid pet type.")
        except sql.Error as e:
            print(f"Error removing pet: {e}")
```

entity:

adoptionevent.py

```
class AdoptionEvent():
    def __init__(self):
        self.personname = ''
        self.eventdetails = ''
```

cashdonation.py

```
from entity.donation import Donation
from datetime import datetime
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):
        # Implement cash donation recording logic here
        print(f"Cash donation of {self.amount} recorded on {self.donation_date}.")
```

cat.py

```
from entity.pet import Pet
from exception.exception import *
class Cat(Pet):
    def __init__(self, name, age, breed, cat_color):
        super().__init__(name, age, breed)
        if not isinstance(cat_color, str):
            raise InvalidNameError("Cat color must be a string")
        for i in cat_color:
            if not i.isalpha() and not i.isspace():
                raise InvalidNameError("Color must be a string and should not contain
numbers")
        self.cat_color = cat_color

    def set_cat_color(self, cat_color):
        if not isinstance(cat_color, str):
            raise InvalidNameError("Cat color must be a string")
        for i in cat_color:
            if not i.isalpha() and not i.isspace():
                raise InvalidNameError("Color must be a string and should not contain
numbers")
        self.cat_color = cat_color

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

dog.py

```
from entity.pet import Pet
from exception.exception import *
class Dog(Pet):
    def __init__(self, name, age, breed, dog_breed):
        super().__init__(name, age, breed)
        if not isinstance(dog_breed, str):
            raise InvalidNameError("Breed should be a string and should not contain
numbers")
        for i in dog_breed:
            if not i.isalpha() and not i.isspace():
                raise InvalidNameError("Breed must be a string and should not contain
numbers")
        self.dog_breed = dog_breed

    def get_dog_breed(self):
        return self.dog_breed

    def set_dog_breed(self, dog_breed):
        if not isinstance(dog_breed, str):
            raise InvalidNameError("Breed should be a string and should not contain
numbers")
        for i in dog_breed:
            if not i.isalpha() and not i.isspace():
                raise InvalidNameError("Breed must be a string and should not contain
numbers")
        self.dog_breed = dog_breed
```

donation.py

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

IAdoptable.py

```
class IAdoptable:
    def Adopt(self):
        pass
```

itemdonation.py

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

pet.py

```
from exception.exception import *
class Pet:
    def __init__(self, name, age, breed):
        if not isinstance(name, str):
            raise InvalidNameError()
        for i in name:
            if not i.isalpha() and not i.isspace():
                raise InvalidNameError()
        if not isinstance(age, int) or age < 0:
            raise InvalidAgeError("Age must be a non-negative integer")
        if not isinstance(breed, str):
            raise InvalidNameError("Breed must be a string and should not contain
numbers")
        for i in breed:
            if not i.isalpha() and not i.isspace():
                raise InvalidNameError("Breed must be a string and should not contain
numbers")
        self.name = name
        self.age = age
        self.breed = breed

    def get_name(self):
        return self.name

    def set_name(self, name):
        if not isinstance(name, str):
            raise InvalidNameError()
        for i in name:
            if not i.isalpha() and not i.isspace():
                raise InvalidNameError()
        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):
        if not isinstance(breed, str):
            raise InvalidNameError("Breed must be a string and should not contain
numbers")
        for i in breed:
            if not i.isalpha() and not i.isspace():
                raise InvalidNameError("Breed must be a string and should not contain
numbers")
        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:
            if not isinstance(new_breed, str):
                raise InvalidNameError("Breed must be a string and should not contain
numbers")
            for i in new_breed:
                if not i.isalpha() and not i.isspace():
                    raise InvalidNameError("Breed must be a string and should not
contain numbers")
            self.breed = new_breed

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


try:

    pet1 = Pet("scoop", 3, 'Dog')
except InvalidNameError as e:
    print(e)
except InvalidAgeError as e:
    print(e)
except Exception as e:
    print(e)
```

petshelter.py

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

exception:

exception.py

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

class DuplicateObjError(Exception):
    def __init__(self, message="Duplicate pet object"):
        self.message = message
        super().__init__(self.message)


class FileHandlingException(Exception):
    def __init__(self, message="This pet is already adopted"):
        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 InvalidAgeError(Exception):
    def __init__(self, message="Invalid age for a dog"):
        self.message = message
        super().__init__(self.message)

class InvalidAmountError(Exception):
    def __init__(self, message="Must be a positive decimal."):
        self.message = message
        super().__init__(self.message)

class InvalidNameError(Exception):
    def __init__(self, message="Name must be a string and should not contain numbers"):
        self.message = message
        super().__init__(self.message)

class NullReferenceException(Exception):
    def __init__(self, message="It is missing some details"):
        self.message = message
        super().__init__(self.message)
```

util:

dbConnection.py

```
import mysql.connector as sql
class dbConnection:
    # THIS FUNCTION WILL GET THE CONNECTION
    def open(self):
        try:
            self.conn = sql.connect(host='localhost', database='petpalscc',
user='root', password='sweetysmiley')
            self.stmt = self.conn.cursor()
        except Exception as e:
            print(str(e) + '-----DATABASE NOT FOUND-----')

    # THIS FUNCTION  WILL CLOSE THE CONNECTION
    def close(self):
        try:
```

```
            self.conn.close()
        except Exception:
            pass
```

main:

main.py

```
from dao.itemdonation import *
from entity.cat import Cat
from entity.dog import Dog
from entity.pet import Pet
from exception.exception import *
from dao.petshelter import *
from dao.cashdonation import *
from dao.adoptionevent import *
from datetime import datetime

try:
    pet1 = Pet("python", 99, 'Animal')
    pet2 = Pet("rabbit", 98, "Animal")
    dog1 = Dog("snoopy", 3, "Dog", "Bull Dog")
    dog2 = Dog("harper", 4, "Dog", "Golden")
    cat1 = Cat("misty", 3, "Cat", "orange")
    cat2 = Cat("sammy", 1, "Cat", "brown")
    petshelter1 = PetShelter()
except InvalidNameError as e:
    print(e)
except InvalidAgeError as e:
    print(e)
except Exception as e:
    print(e)

status = True
Once = True
try:
    petshelter1 = PetShelter()
    cashdonation1 = CashDonation()
    itemdonation1 = ItemDonation()
    adoptionevent1 = AdoptionEvent()
    while Once:
        petshelter1.create_table()
        cashdonation1.createTable()
        adoptionevent1.create_event()
        adoptionevent1.create_participants()
        adoptionevent1.create_adoption()
        itemdonation1.createTable()
        Once = False

    while status:

        print("Below are the list of applications")
        print("\n1.addCustomer\t2.list available pets\n3.Delete Pet\t"
              "4.Record Amount Donation\n5.View Amount DonationData\t6.Record Toys
Donation\n7.view "
              "Toys Donation data\t8.Register Participant for event"
              "\n9.Get Participants Details \t10.Add New Host Event\n11.Get Host 7Event
Details\t12.See Adoption Data "
              "\n13.Proceed for Adoption\t14.Exit")
        choice = int(input("enter above choices"))

        if choice == 1:
```

```
            print("create object to add into Pets table")
            print("1.For Pets\n2.For Dogs\n3.For Cat")
            choice = int(input("enter choice"))
            if choice == 1:
                name = input("enter name")
                age = int(input("enter age"))
                breed = input("enter breeed")
                obj1 = Pet(name, age, breed)
                petshelter1.AddPet(obj1)
            elif choice == 2:
                name = input("enter name")
                age = int(input("enter age"))
                breed = input("enter breeed")
                dog_breed = input("enter sub breed")
                obj1 = Dog(name, age, breed, dog_breed)
                petshelter1.AddPet(obj1)
            else:
                name = input("enter name")
                age = int(input("enter age"))
                breed = input("enter breed")
                color = input("enter color")
                obj1 = Cat(name, age, breed, color)
                petshelter1.AddPet(obj1)
        elif choice == 2:
            petshelter1.ListAvailablePets()
        elif choice == 3:
            id = int(input("enter ID"))
            petshelter1.RemovePet(id)
        elif choice == 4:
            name = input("enter name")
            amount = int(input("enter amount"))
            date = input("enter date")
            cashdonation1 = CashDonation(name, amount, date)
            cashdonation1.RecordDonation()
        elif choice == 5:
            cashdonation1.ViewAmountDonationData()
        elif choice == 6:
            name = input("enter name")
            amount = int(input("enter amount"))
            item = input("enter item")
            itemdonation1 = ItemDonation(name, amount, item)
            itemdonation1.RecordDonation()
        elif choice == 7:
            itemdonation1.ViewItemDonationData()
        elif choice == 8:
            adoptionevent1.RegisterParticipant()
        elif choice == 9:
            adoptionevent1.GetParticipants()
        elif choice == 10:
            adoptionevent1.HostEvent()
        elif choice == 11:
            adoptionevent1.GetEvent()
        elif choice ==12:
            adoptionevent1.ViewAdoption()
        elif choice == 13:
            adoptionevent1.Adopt()
        else:

            status = False
except InvalidNameError as e:
    print(e)
except InvalidAgeError as e:
    print(e)
except InvalidAmountError as e:
    print(e)
except InsufficientFundsException as e:
    print(e)
```

```
except DuplicateObjError as e:
    print(e)
except NullReferenceException as e:
    print(e)
except AdoptionException as e:
    print(e)
except FileHandlingException as e:
    print(e)
except Exception as e:
    print(e)
```

outputs:

```
Table Pets is created.
CashDonation table is created.
Event table is created.
Participants table is created.
Adopt table is created.
ItemDonation table is created.
Below are the list of applications

1.addCustomer   2.list available pets
3.Delete Pet    4.Record Amount Donation
5.View Amount DonationData  6.Record Toys Donation
7.view Toys Donation data   8.Register Participant for event
9.Get Participants Details  10.Add New Host Event
11.Get Host 7Event Details  12.See Adoption Data
13.Proceed for Adoption 14.Exit
enter above choices
```

```
enter above choices1
create object to add into Pets table
1.For Pets
2.For Dogs
3.For Cat
enter choice1
enter namepig
enter age5
enter breeedlocal
(1, 'pig', 5, 'local')
pig added to the list of available pets.
```

```
enter above choices2
(1, 'pig', 5, 'local')
```

```
enter above choices3
enter ID1
deleted successfully!!
```

```
enter above choices4
enter nameneeha
enter amount1000
enter date2024-03-05
Cash donation recorded successfully.
```

```
enter above choices5
(1, 'neeha', Decimal('1000.00'))
```

```
enter above choices6
enter nameJay
enter amount1500
enter itemfood
Record added to ItemDonation table.
```

```
enter above choices7
ItemDonation table data:
ID: 1, Donor Name: Jay, Amount: 1500.00, ItemType: food
```

```
enter above choices8
Enter participant name: Harry
Participant 'Harry' added successfully.
```

```
enter above choices9
(1, 'Harry')
```

```
enter above choices10
Enter event details: pets show
Event hosted successfully.
```

```
enter above choices11
(1, 'pets show')
```