Coding Challenge: PetPals, The Pet Adoption Platform

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

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:

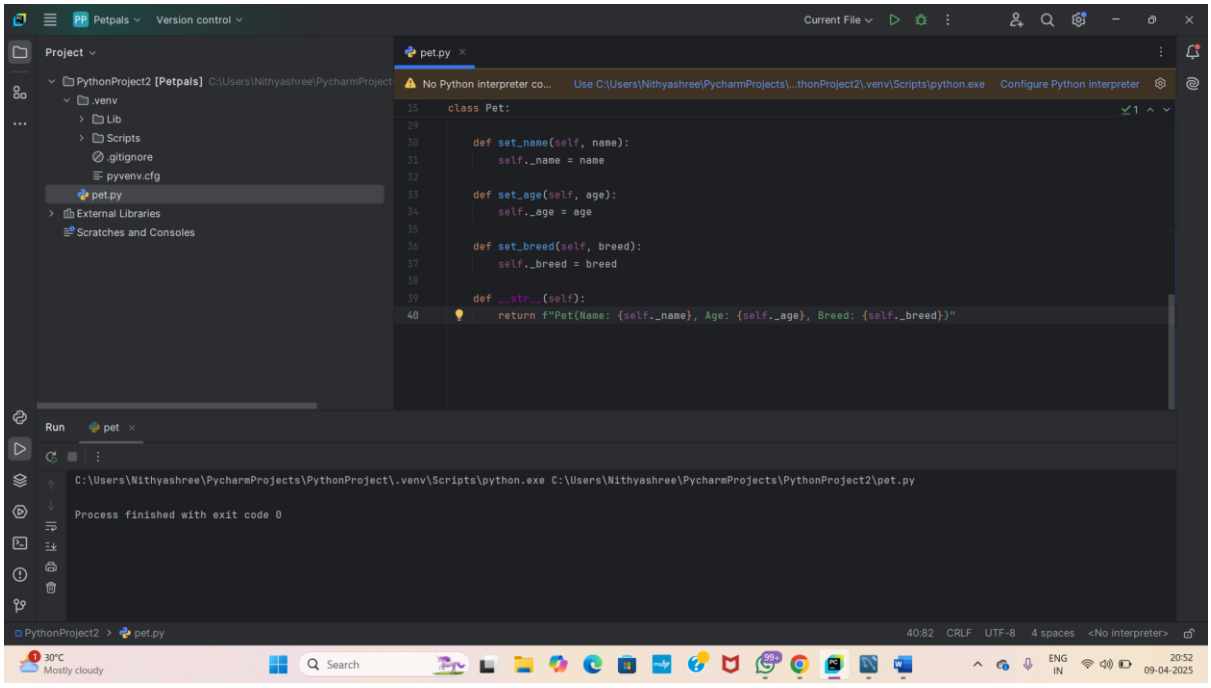
• Constructor to initialize Name, Age, and Breed.

• Getters and setters for attributes.

• ToString() method to provide a string representation of the pet.

```
from abc import ABC, abstractmethod
from datetime import datetime
import mysql.connector

def get_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",      # change this
        password="Deepa@25",   # change this
        database="petpals"         # make sure this matches your DB
    )


# -------------------- Base Pet Class ---------
class Pet:
    def __init__(self, name: str, age: int, breed: str):
        self._name = name
        self._age = age
        self._breed = breed

    def get_name(self):
```

```
        return self._name

    def get_age(self):
        return self._age

    def get_breed(self):
        return self._breed

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

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

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

    def __str__(self):
        return f"Pet(Name: {self._name}, Age: {self._age}, Breed: {self._breed})"
```



Dog Class (Inherits from Pet):

Additional Attributes:

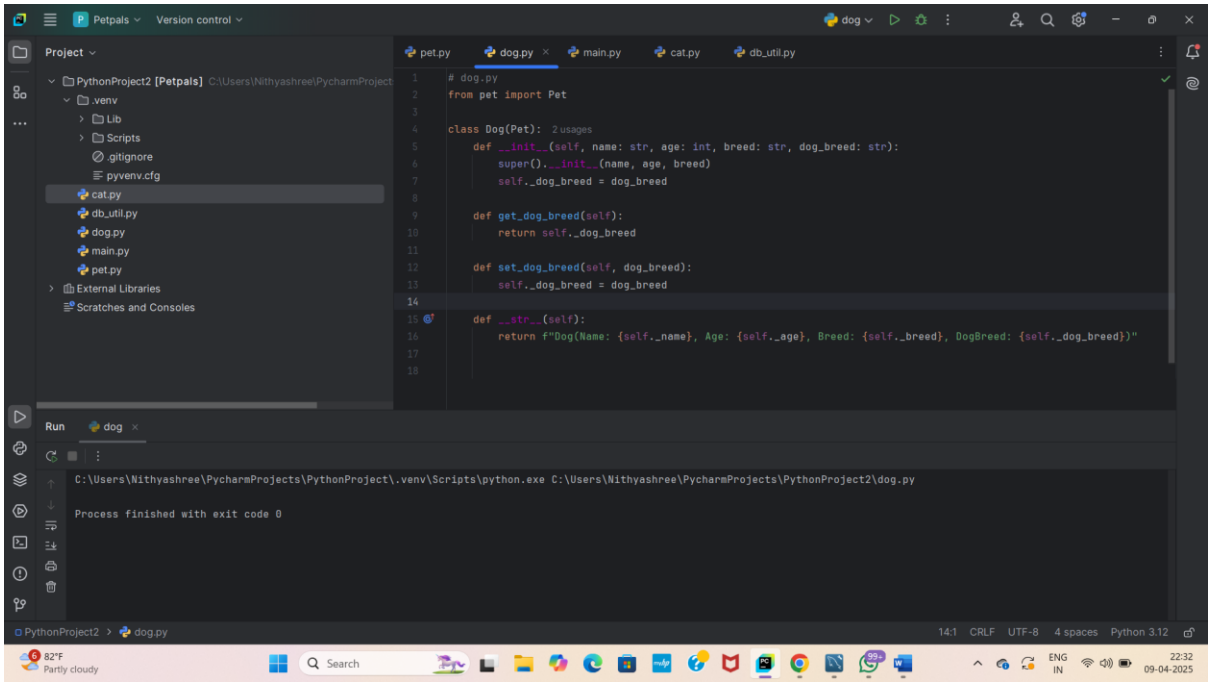
• DogBreed (string): The specific breed of the dog.

Additional Methods:

• Constructor to initialize DogBreed.

• Getters and setters for DogBreed.

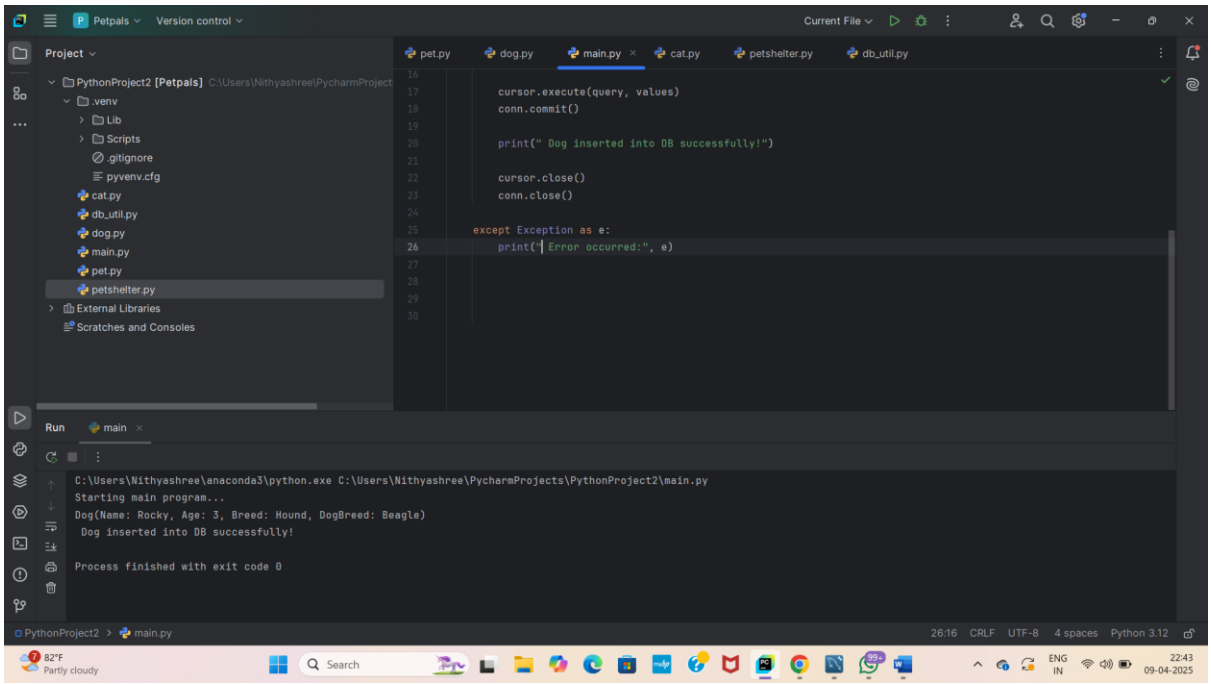
```
# dog.py
from pet import Pet

class Dog(Pet):
    def __init__(self, name: str, age: int, breed: str, dog_breed: str):
        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 __str__(self):
        return f"Dog(Name: {self._name}, Age: {self._age}, Breed: {self._breed}, DogBreed:
{self._dog_breed})"
```



EXAMPLE USAGE IN main.py:

```
from dog import Dog

from db_util import get_connection


if __name__ == "__main__":

    print("Starting main program...")


    dog = Dog("Rocky", 3, "Hound", "Beagle")
```

```
print(dog)

try:
    conn = get_connection()
    cursor = conn.cursor()

    query = "INSERT INTO pets (Name, Age, Breed) VALUES (%s, %s, %s)"
    values = (dog.get_name(), dog.get_age(), dog.get_breed())

    cursor.execute(query, values)
    conn.commit()

    print(" Dog inserted into DB successfully!")

    cursor.close()
    conn.close()

except Exception as e:
    print(" Error occurred:", e)
```



Cat Class (Inherits from Pet):

Additional Attributes:

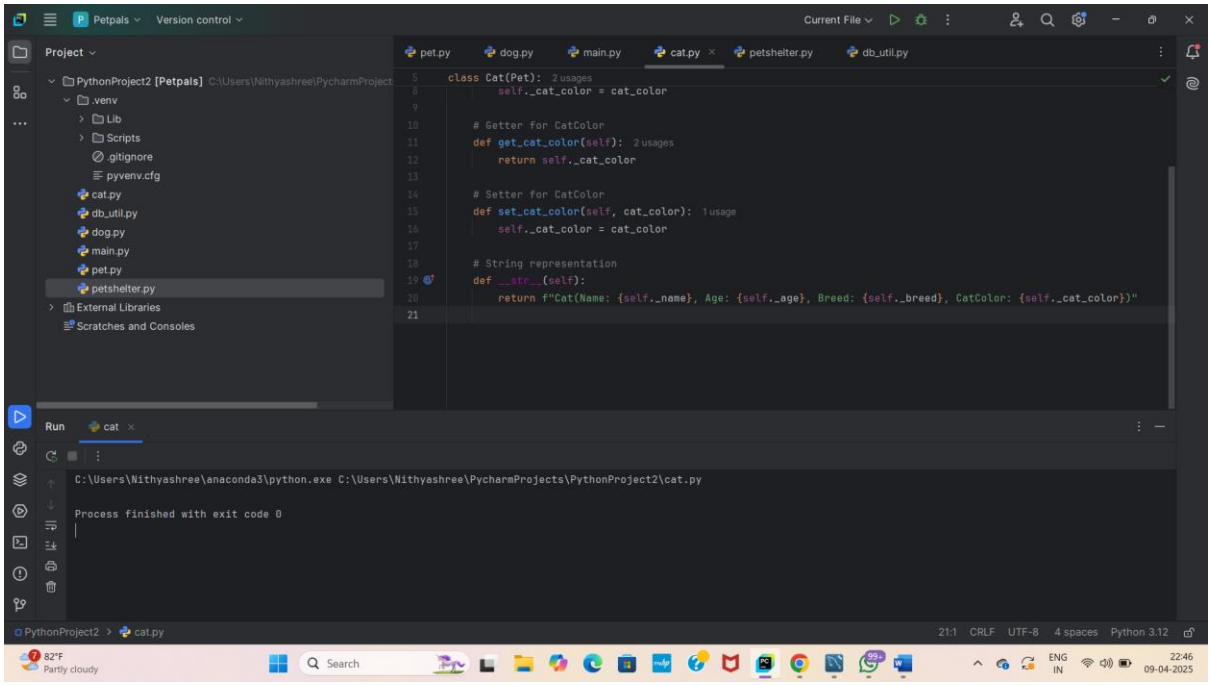
• CatColor (string): The color of the cat.

Additional Methods:

• Constructor to initialize CatColor.

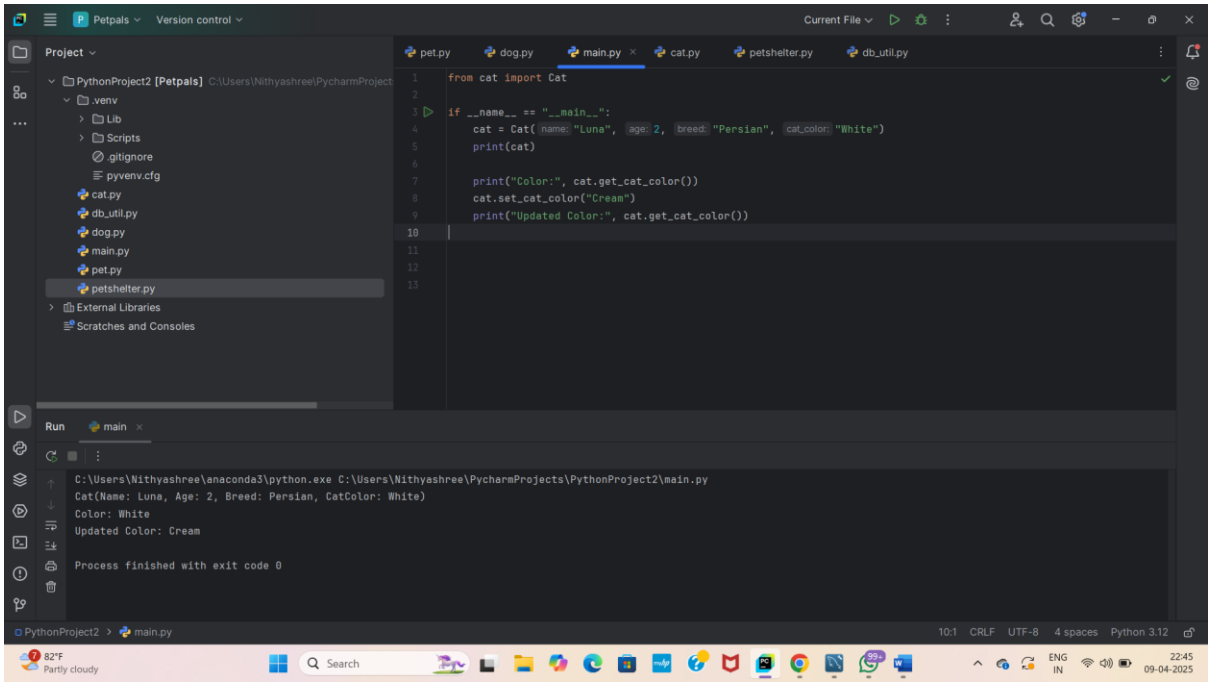
• Getters and setters for CatColor

```
# cat.py

from pet import Pet

class Cat(Pet):
    def __init__(self, name: str, age: int, breed: str, cat_color: str):
        super().__init__(name, age, breed)
        self._cat_color = cat_color

    # Getter for CatColor
    def get_cat_color(self):
        return self._cat_color

    # Setter for CatColor
    def set_cat_color(self, cat_color):
        self._cat_color = cat_color

    # String representation
    def __str__(self):
        return f"Cat(Name: {self._name}, Age: {self._age}, Breed: {self._breed}, CatColor:
{self._cat_color})"
```



EXAMPLE USAGE IN main.py:

```
from cat import Cat


if __name__ == "__main__":
    cat = Cat("Luna", 2, "Persian", "White")
    print(cat)


    print("Color:", cat.get_cat_color())
    cat.set_cat_color("Cream")
    print("Updated Color:", cat.get_cat_color())
```



3.PetShelter Class:

Attributes:

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

Methods:

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

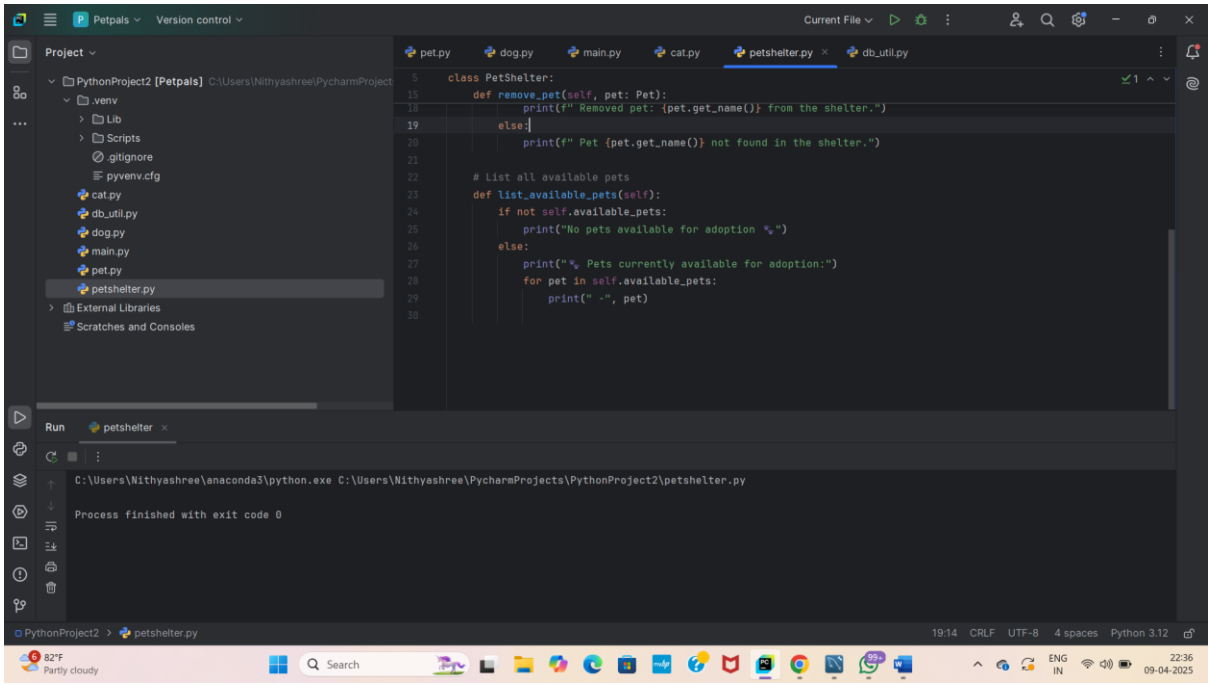
• RemovePet(Pet pet): Removes a pet from the list of available pets.

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

petshelter.py

```
from pet import Pet

class PetShelter:
    def __init__(self):
        self.available_pets = []  # List to store Pet objects

    # Add a pet to the shelter
    def add_pet(self, pet: Pet):
        self.available_pets.append(pet)
        print(f" Added pet: {pet.get_name()} to the shelter.")

    # Remove a pet from the shelter
    def remove_pet(self, pet: Pet):
        if pet in self.available_pets:
            self.available_pets.remove(pet)
            print(f" Removed pet: {pet.get_name()} from the shelter.")
        else:
```

```
        print(f" Pet {pet.get_name()} not found in the shelter.")

    # List all available pets
    def list_available_pets(self):
        if not self.available_pets:
            print("No pets available for adoption ")
        else:
            print(" Pets currently available for adoption:")
            for pet in self.available_pets:
                print(" -", pet)
```



EXAMPLE USAGE IN main.py:

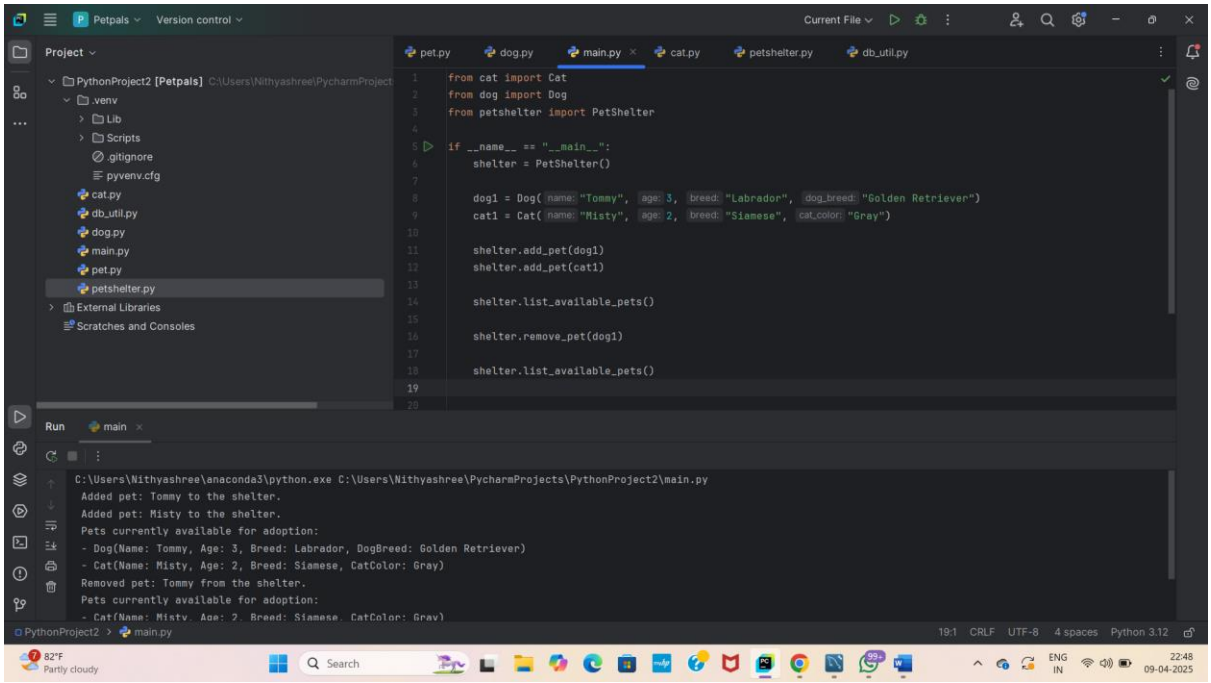
```
from cat import Cat

from dog import Dog

from petshelter import PetShelter


if __name__ == "__main__":

    shelter = PetShelter()


    dog1 = Dog("Tommy", 3, "Labrador", "Golden Retriever")

    cat1 = Cat("Misty", 2, "Siamese", "Gray")


    shelter.add_pet(dog1)
```

shelter.add_pet(cat1)

shelter.list_available_pets()

shelter.remove_pet(dog1)

shelter.list_available_pets()



4.Donation Class (Abstract):

Attributes:

• DonorName (string): The name of the donor.

• Amount (decimal): The donation amount.

Methods:

• Constructor to initialize DonorName and Amount.

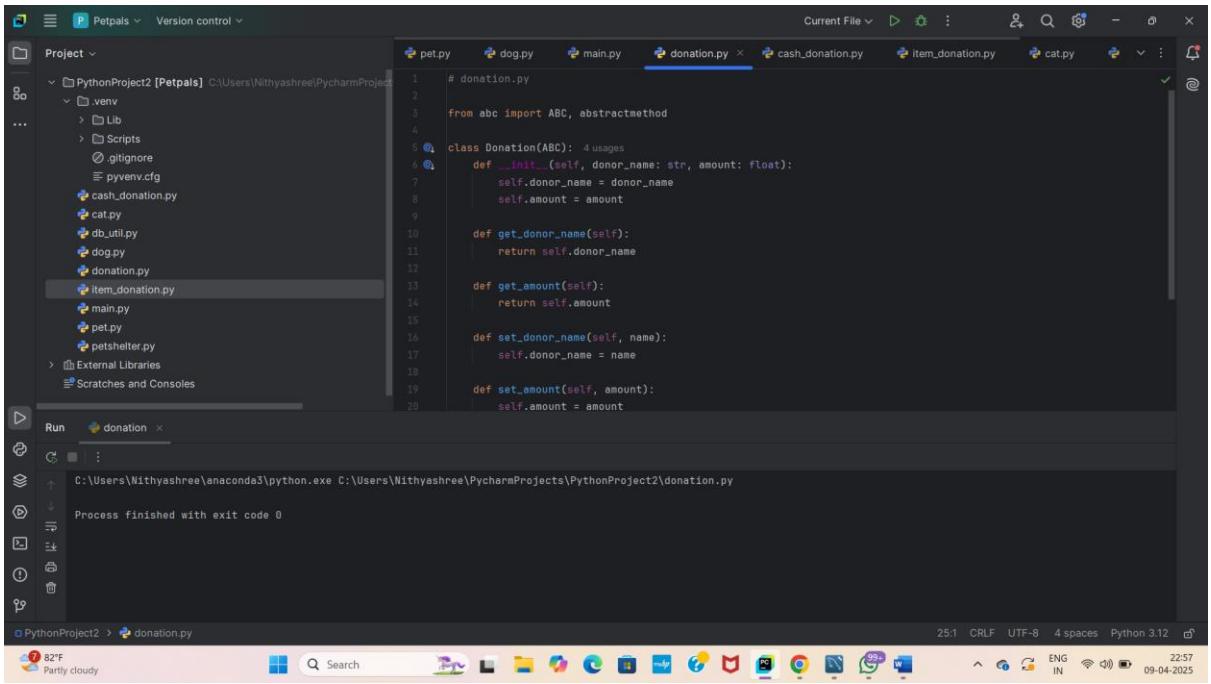
• Abstract method RecordDonation() to record the donation (to be implemented in derived classes)

donation.py

from abc import ABC, abstractmethod

class Donation(ABC):

```
def __init__(self, donor_name: str, amount: float):
    self.donor_name = donor_name
    self.amount = amount

def get_donor_name(self):
    return self.donor_name

def get_amount(self):
    return self.amount

def set_donor_name(self, name):
    self.donor_name = name

def set_amount(self, amount):
    self.amount = amount

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



CashDonation Class (Derived from Donation):

Additional Attributes:

• DonationDate (DateTime): The date of the cash donation.

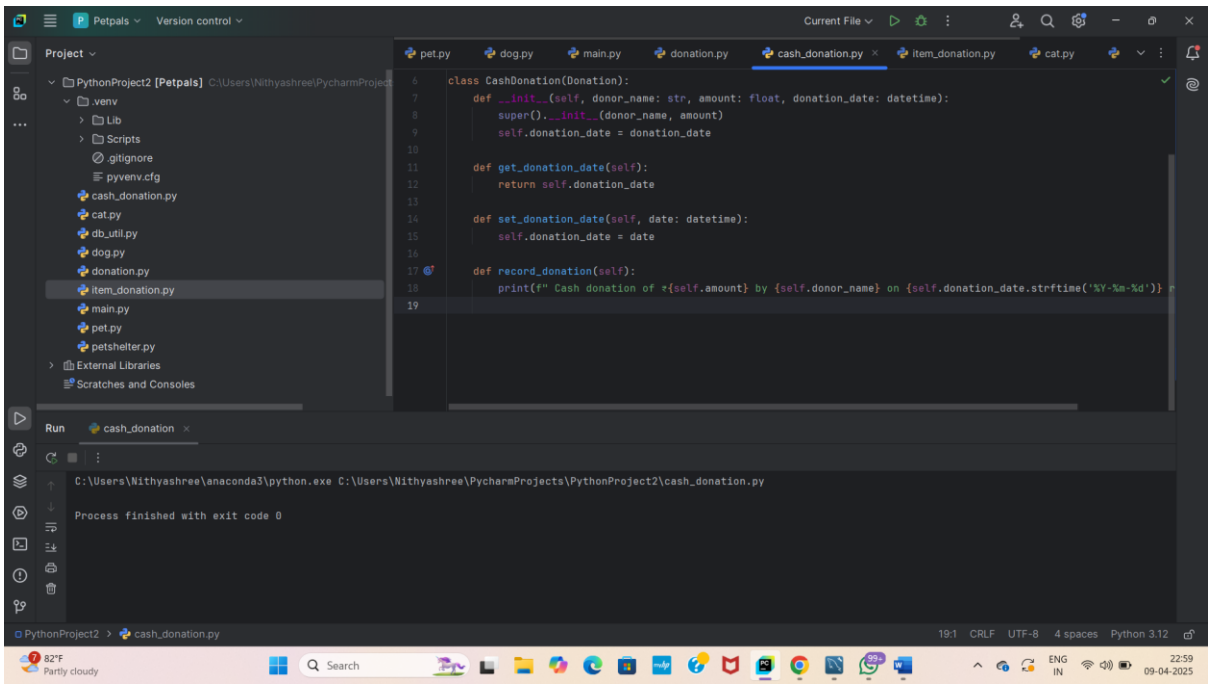
Additional Methods:

• Constructor to initialize DonationDate.

• Implementation of RecordDonation() to record a cash donation.

cash_donation.py

```
from donation import Donation
from datetime import datetime

class CashDonation(Donation):
    def __init__(self, donor_name: str, amount: float, donation_date: datetime):
        super().__init__(donor_name, amount)
        self.donation_date = donation_date

    def get_donation_date(self):
        return self.donation_date

    def set_donation_date(self, date: datetime):
        self.donation_date = date

    def record_donation(self):
        print(f" Cash donation of ₹{self.amount} by {self.donor_name} on
{self.donation_date.strftime('%Y-%m-%d')} recorded.")
```



ItemDonation Class (Derived from Donation):

Additional Attributes:

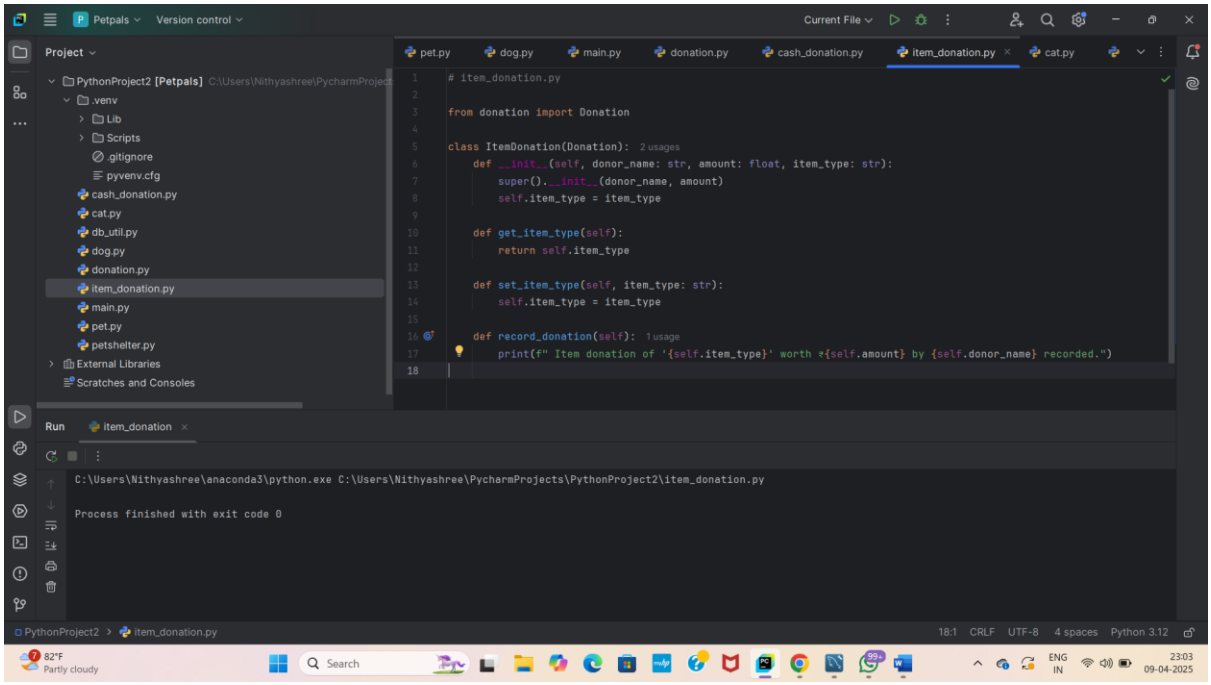
• ItemType (string): The type of item donated (e.g., food, toys).

Additional Methods:

• Constructor to initialize ItemType.

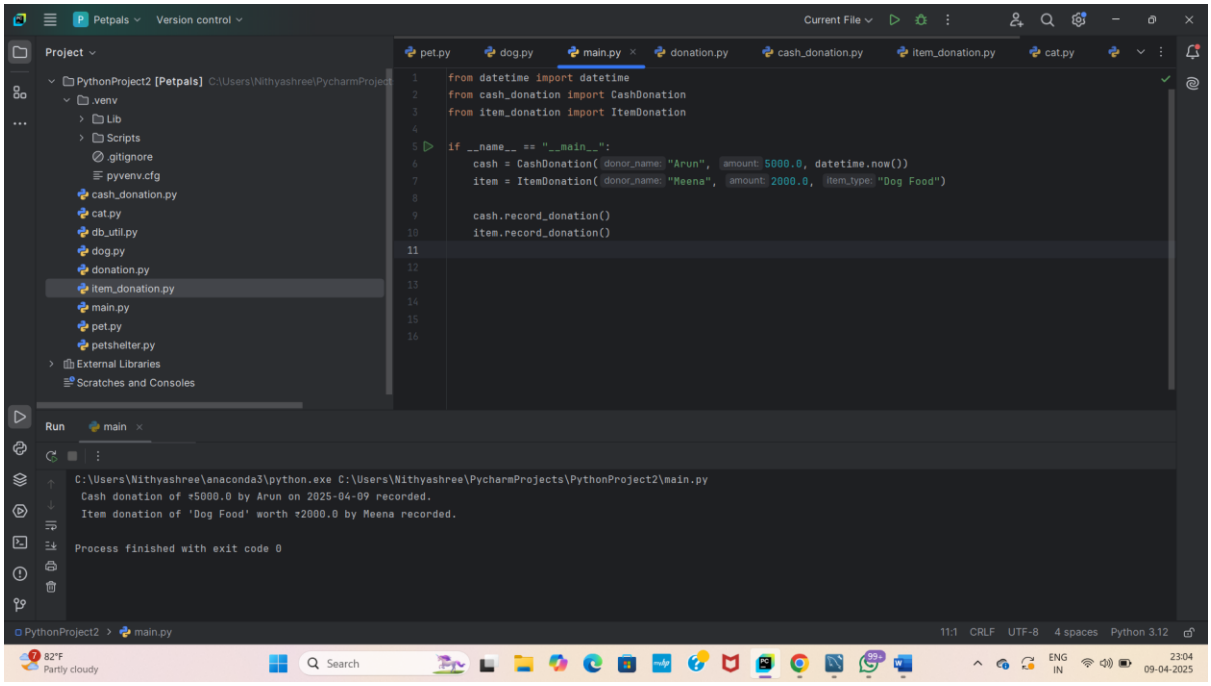
• Implementation of RecordDonation() to record an item donation

```
# item_donation.py

from donation import Donation

class ItemDonation(Donation):
    def __init__(self, donor_name: str, amount: float, item_type: str):
        super().__init__(donor_name, amount)
        self.item_type = item_type

    def get_item_type(self):
        return self.item_type

    def set_item_type(self, item_type: str):
        self.item_type = item_type

    def record_donation(self):
        print(f" Item donation of '{self.item_type}' worth ₹{self.amount} by {self.donor_name}
recorded.")
```



EXAMPLE USAGE IN main.py:

```
from datetime import datetime

from cash_donation import CashDonation

from item_donation import ItemDonation


if __name__ == "__main__":

    cash = CashDonation("Arun", 5000.0, datetime.now())

    item = ItemDonation("Meena", 2000.0, "Dog Food")


    cash.record_donation()

    item.record_donation()
```



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:

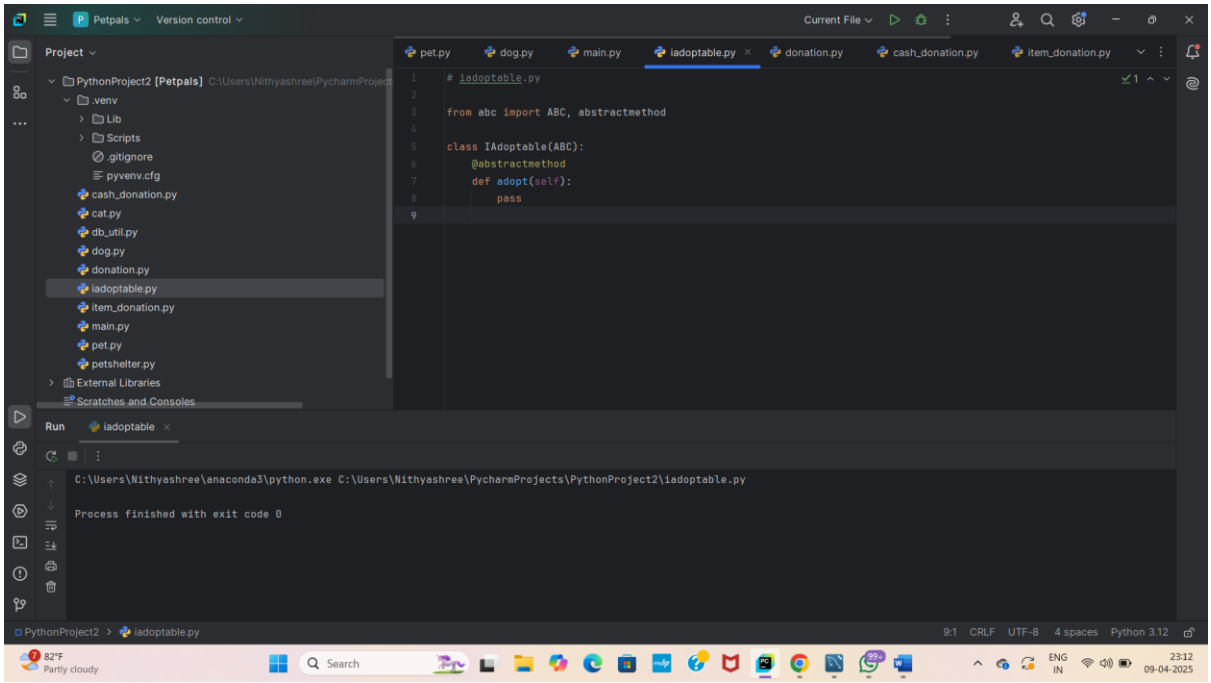
• HostEvent(): Hosts the adoption event.

• RegisterParticipant(IAdoptable participant): Registers a participant for the event

IAdoptable Interface:

```
# iadoptable.py
```

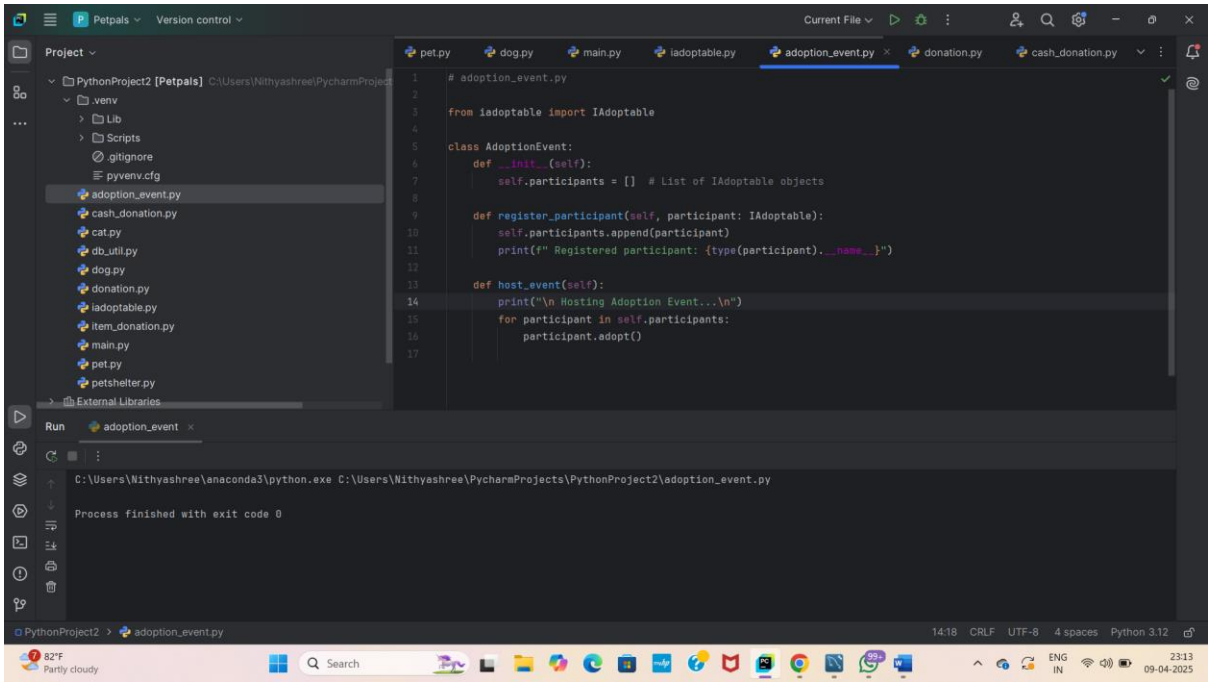
```
from abc import ABC, abstractmethod


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



AdoptionEvent Class:

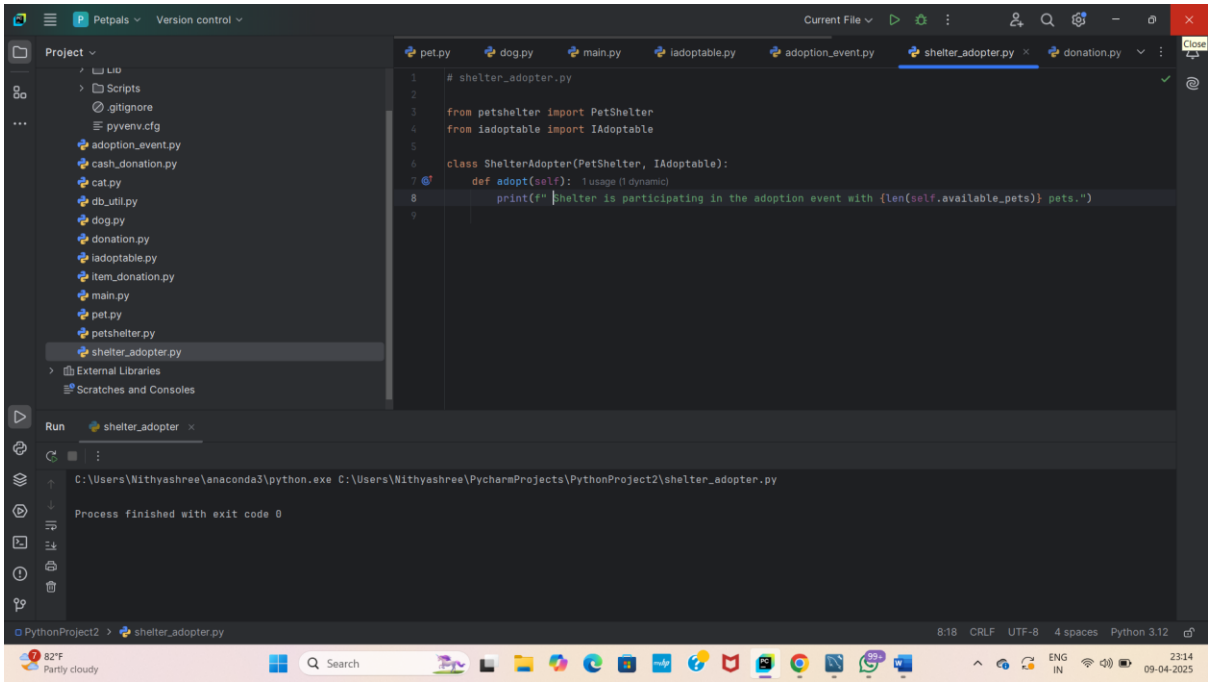
```
# adoption_event.py


from iadoptable import IAdoptable


class AdoptionEvent:
    def __init__(self):
        self.participants = []  # List of IAdoptable objects


    def register_participant(self, participant: IAdoptable):
        self.participants.append(participant)
        print(f" Registered participant: {type(participant).__name__}")
```

```
def host_event(self):
    print("\n Hosting Adoption Event...\n")
    for participant in self.participants:
        participant.adopt()
```



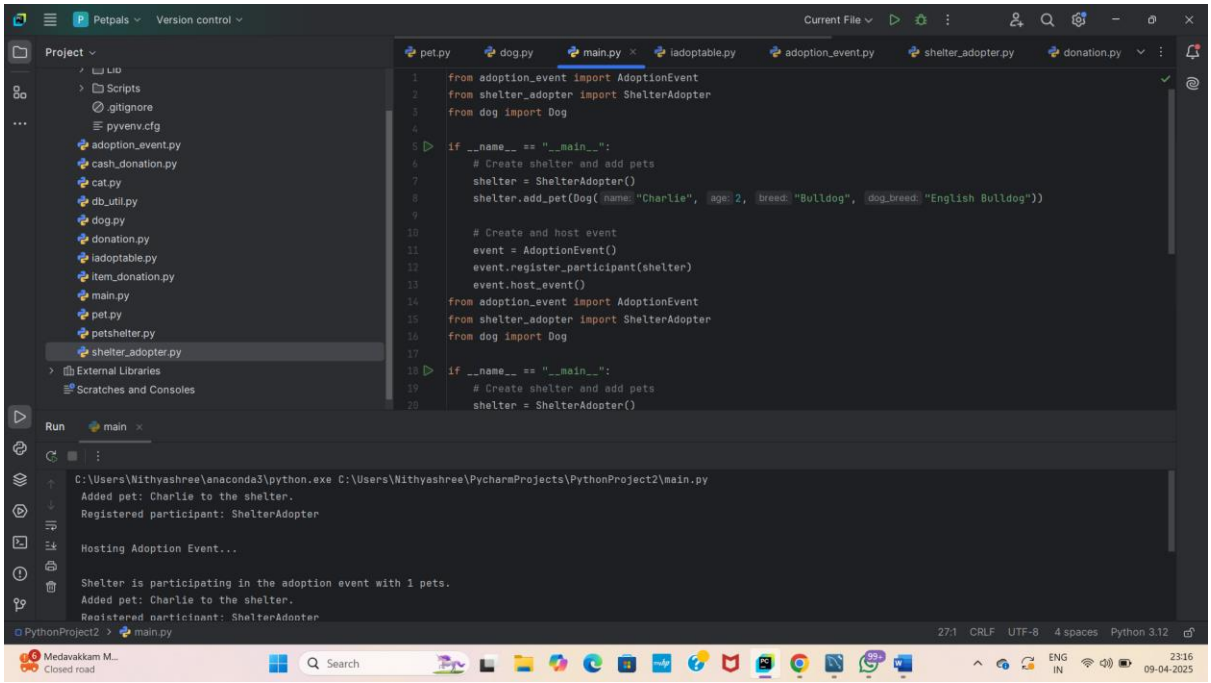
ShelterAdopter class:

```
# shelter_adopter.py


from petshelter import PetShelter
from iadoptable import IAdoptable


class ShelterAdopter(PetShelter, IAdoptable):
    def adopt(self):
        print(f" Shelter is participating in the adoption event with {len(self.available_pets)} pets.")
```



EXAMPLE USAGE IN main.py:

```
from adoption_event import AdoptionEvent
from shelter_adopter import ShelterAdopter
from dog import Dog


if __name__ == "__main__":
    # Create shelter and add pets
    shelter = ShelterAdopter()
    shelter.add_pet(Dog("Charlie", 2, "Bulldog", "English Bulldog"))


    # Create and host event
    event = AdoptionEvent()
    event.register_participant(shelter)
    event.host_event()
```



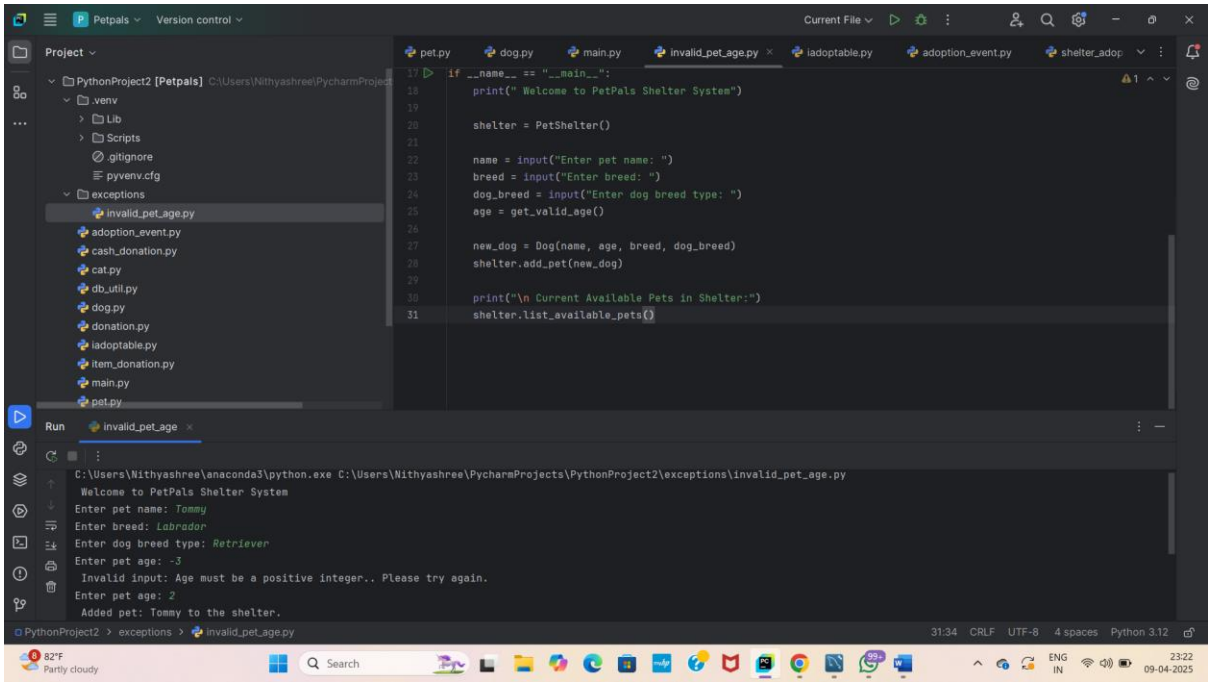
6.Exceptions handling:

• 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.

#invalid_pet_age.py

```
from petshelter import PetShelter
from dog import Dog

def get_valid_age():
    while True:
        try:
            age_input = input("Enter pet age: ")
            age = int(age_input)
            if age <= 0:
                raise ValueError("Age must be a positive integer.")
            return age
        except ValueError as ve:
            print(f" Invalid input: {ve}. Please try again.")

if __name__ == "__main__":
    print(" Welcome to PetPals Shelter System")

    shelter = PetShelter()

    name = input("Enter pet name: ")
```

```
breed = input("Enter breed: ")
dog_breed = input("Enter dog breed type: ")
age = get_valid_age()

new_dog = Dog(name, age, breed, dog_breed)
shelter.add_pet(new_dog)

print("\n Current Available Pets in Shelter:")
shelter.list_available_pets()
```

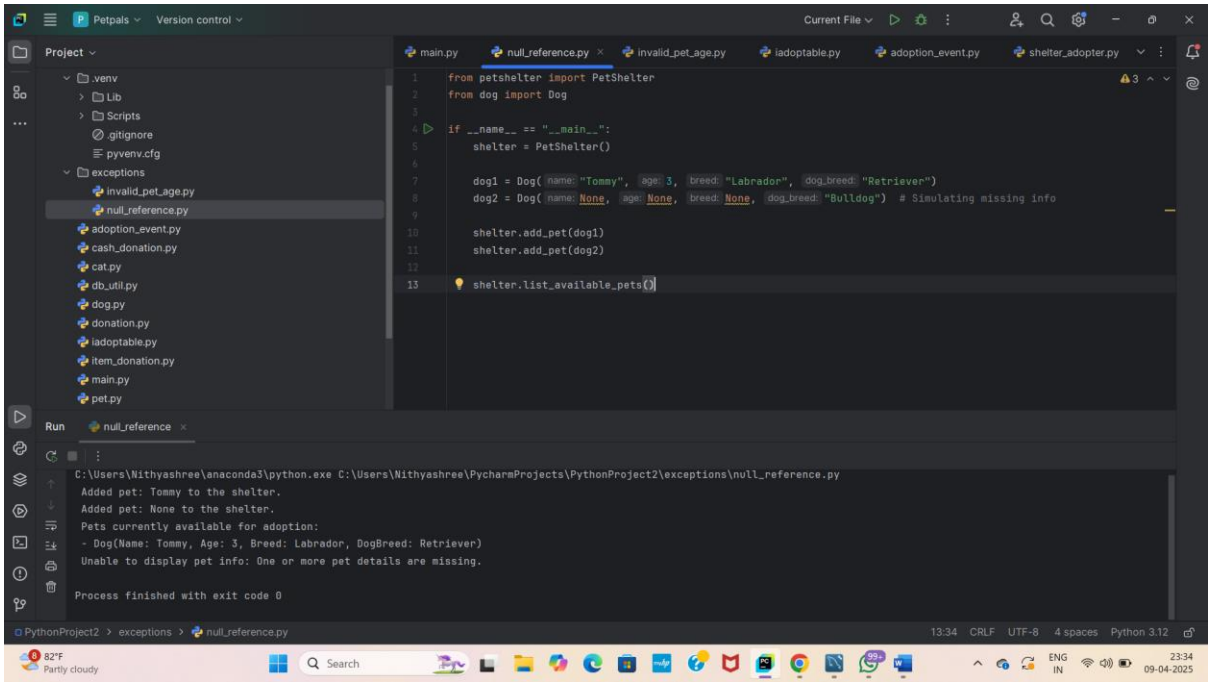


• Null Reference Exception Handling: o In the Pet Adoption Platform, when displaying the list of available pets in a shelter, it's important to handle situations where a pet's properties (e.g., Name, Age) might be null. Implement exception handling to catch null reference exceptions when accessing properties of pets in the shelter and display a message indicating that the information is missing

Updated list_available_pets() in PetShelter Class:

```
# petshelter.py


from pet import Pet  # if needed


class PetShelter:

    def __init__(self):

        self.available_pets = []
```

```
def add_pet(self, pet: Pet):
    self.available_pets.append(pet)
    print(f" Added pet: {pet.get_name()} to the shelter.")

def remove_pet(self, pet: Pet):
    if pet in self.available_pets:
        self.available_pets.remove(pet)
        print(f" Removed pet: {pet.get_name()} from the shelter.")
    else:
        print(f" Pet {pet.get_name()} not found in the shelter.")

def list_available_pets(self):
    if not self.available_pets:
        print("No pets available for adoption ")
    else:
        print(" Pets currently available for adoption:")
        for pet in self.available_pets:
            try:
                if pet.get_name() is None or pet.get_age() is None or pet.get_breed() is None:
                    raise ValueError("One or more pet details are missing.")
                print(" -", pet)
            except Exception as e:
                print(f" Unable to display pet info: {e}")
```



• 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.

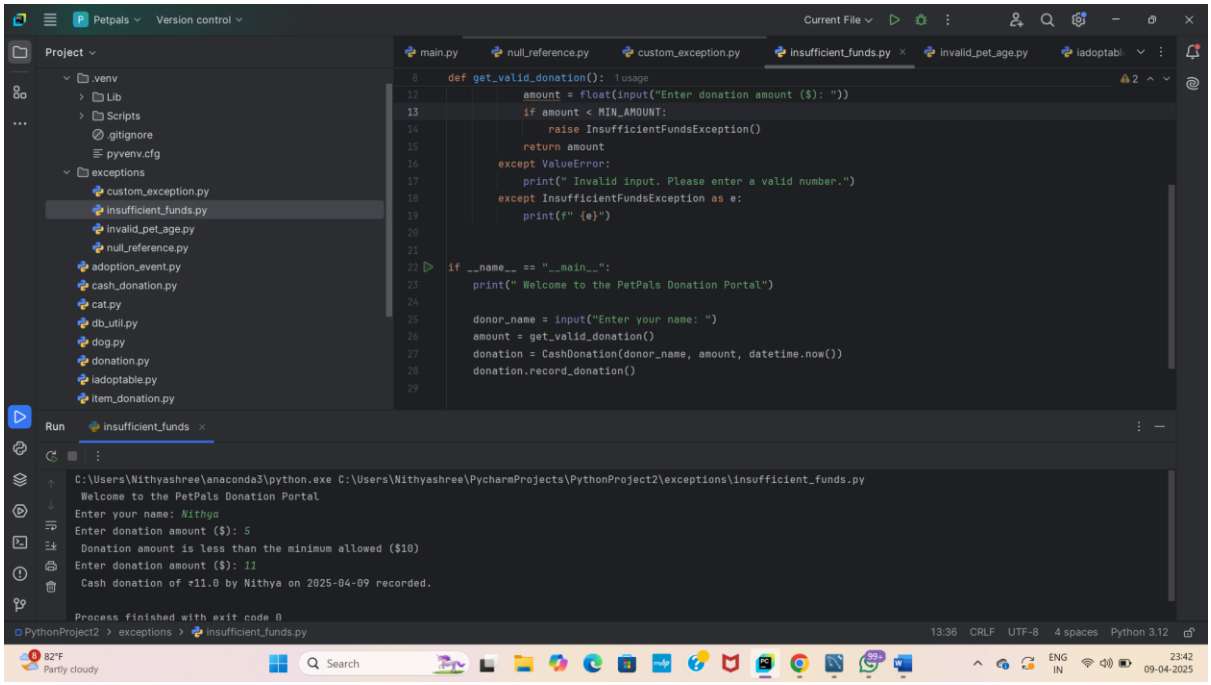
Defining a Custom Exception:

```
# custom_exception.py


class InsufficientFundsException(Exception):

    def __init__(self, message="Donation amount is less than the minimum allowed ($10)"):

        super().__init__(message)
```

Implementing Donation Input with Exception Handling:

```
from cash_donation import CashDonation
from datetime import datetime
from custom_exception import InsufficientFundsException


def get_valid_donation():
    MIN_AMOUNT = 10.0
    while True:
        try:
            amount = float(input("Enter donation amount ($): "))
            if amount < MIN_AMOUNT:
                raise InsufficientFundsException()
```

```
            return amount
        except ValueError:
            print(" Invalid input. Please enter a valid number.")
        except InsufficientFundsException as e:
            print(f" {e}")


if __name__ == "__main__":
    print(" Welcome to the PetPals Donation Portal")

    donor_name = input("Enter your name: ")
    amount = get_valid_donation()
    donation = CashDonation(donor_name, amount, datetime.now())
    donation.record_donation()
```



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

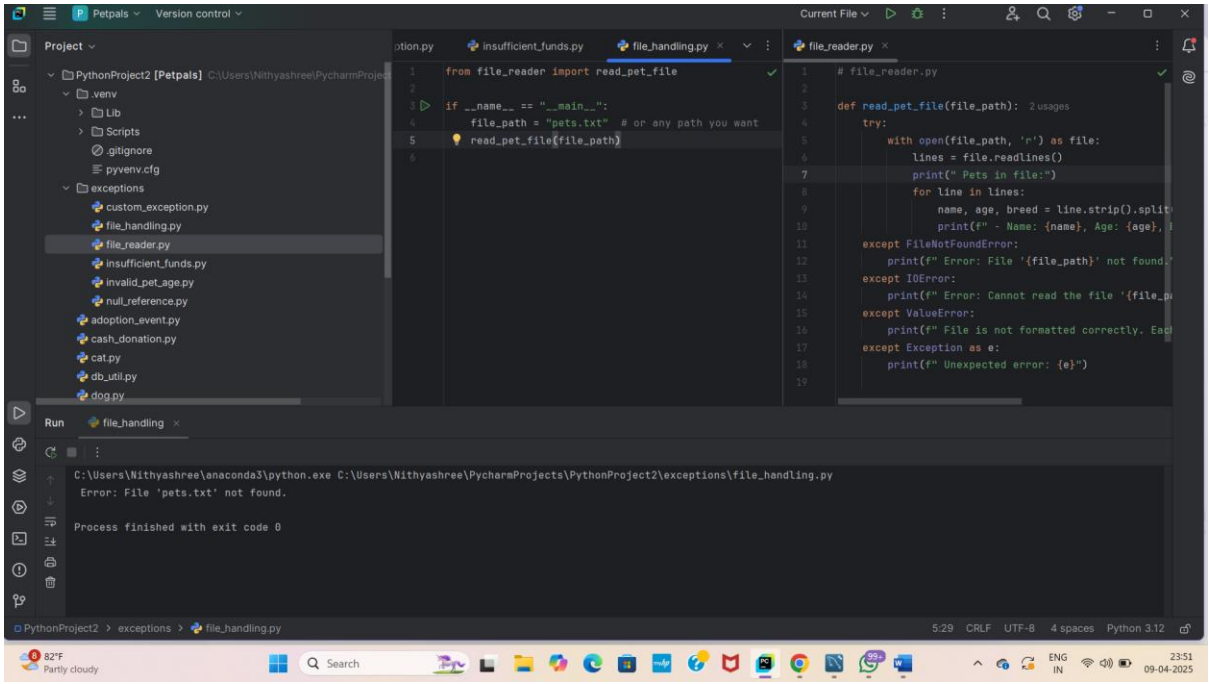
Code to Read the File with Exception Handling:

```
# file_reader.py


def read_pet_file(file_path):

    try:

        with open(file_path, 'r') as file:

            lines = file.readlines()
```

```
            print(" Pets in file:")
            for line in lines:
                name, age, breed = line.strip().split(',')
                print(f" - Name: {name}, Age: {age}, Breed: {breed}")
    except FileNotFoundError:
        print(f" Error: File '{file_path}' not found.")
    except IOError:
        print(f" Error: Cannot read the file '{file_path}'.")
    except ValueError:
        print(f" File is not formatted correctly. Each line should be: Name,Age,Breed")
    except Exception as e:
        print(f" Unexpected error: {e}")
```

file_handling.py in exceptions directory:

```
from file_reader import read_pet_file


if __name__ == "__main__":
    file_path = "pets.txt"  # or any path you want
    read_pet_file(file_path)
```



- **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

Defining the Custom Exception Class:

```
# custom_exception.py


class InsufficientFundsException(Exception):
    def __init__(self, message="Donation amount is less than the minimum allowed ($10)"):
        super().__init__(message)


class AdoptionException(Exception):
    def __init__(self, message="Adoption error occurred"):
        super().__init__(message)
```

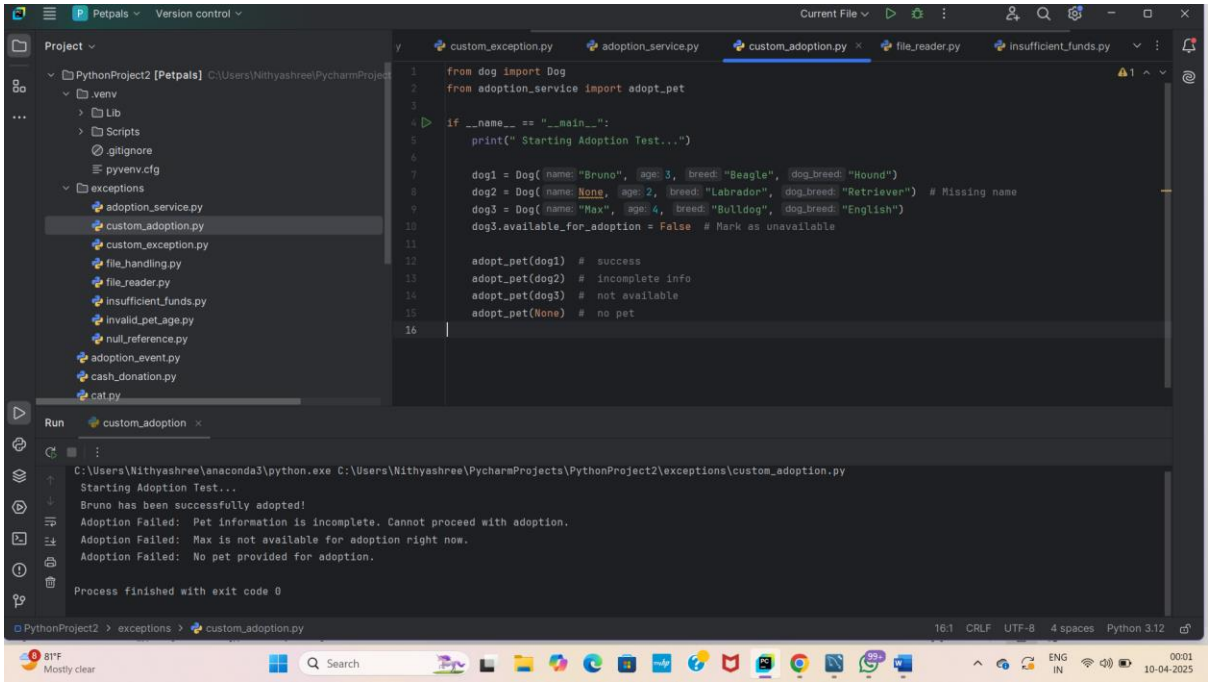
Using AdoptionException:

```
# adoption_service.py


from custom_exception import AdoptionException


def adopt_pet(pet):
    try:
        if pet is None:
            raise AdoptionException(" No pet provided for adoption.")
        if pet.get_name() is None or pet.get_age() is None or pet.get_breed() is None:
            raise AdoptionException(" Pet information is incomplete. Cannot proceed with adoption.")


        # Let's assume pet has a flag .available_for_adoption (just for logic)
        if hasattr(pet, 'available_for_adoption') and not pet.available_for_adoption:
            raise AdoptionException(f" {pet.get_name()} is not available for adoption right now.")


        print(f" {pet.get_name()} has been successfully adopted!")
```

except AdoptionException as ae:

print(f" Adoption Failed: {ae}")

custom_adoption.py in exceptions directory:



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.

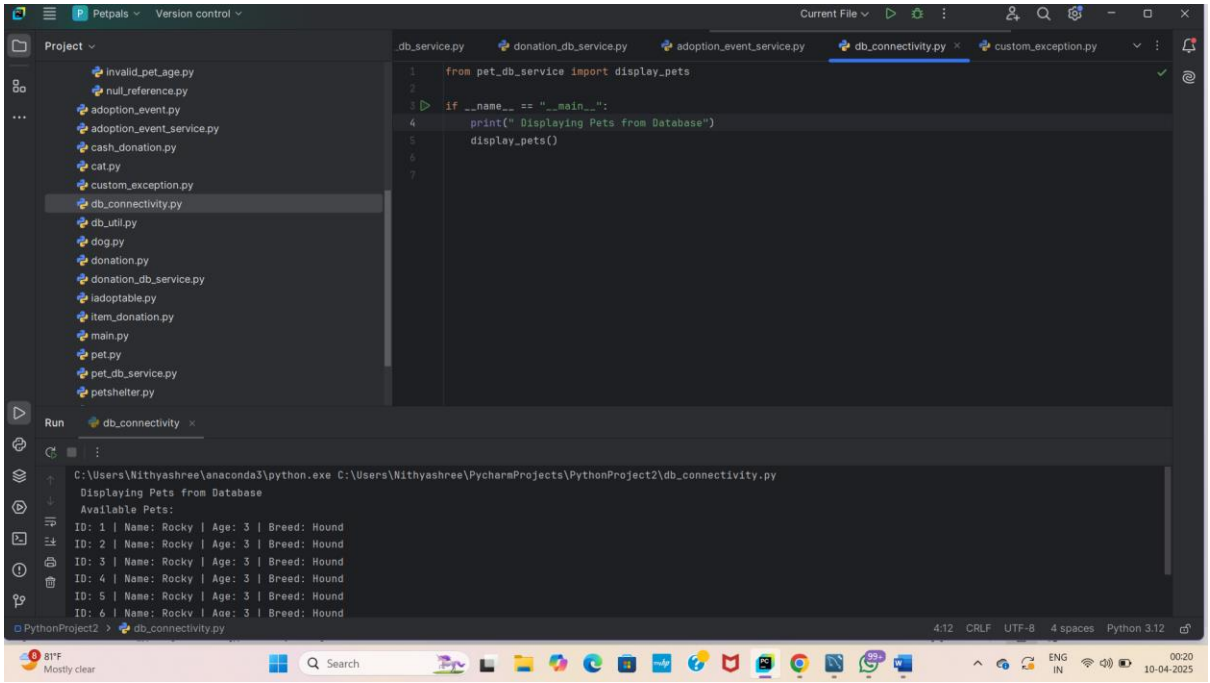
pet_db_service.py:

from pet_db_service import display_pets

if __name__ == "__main__":

print(" Displaying Pets from Database")

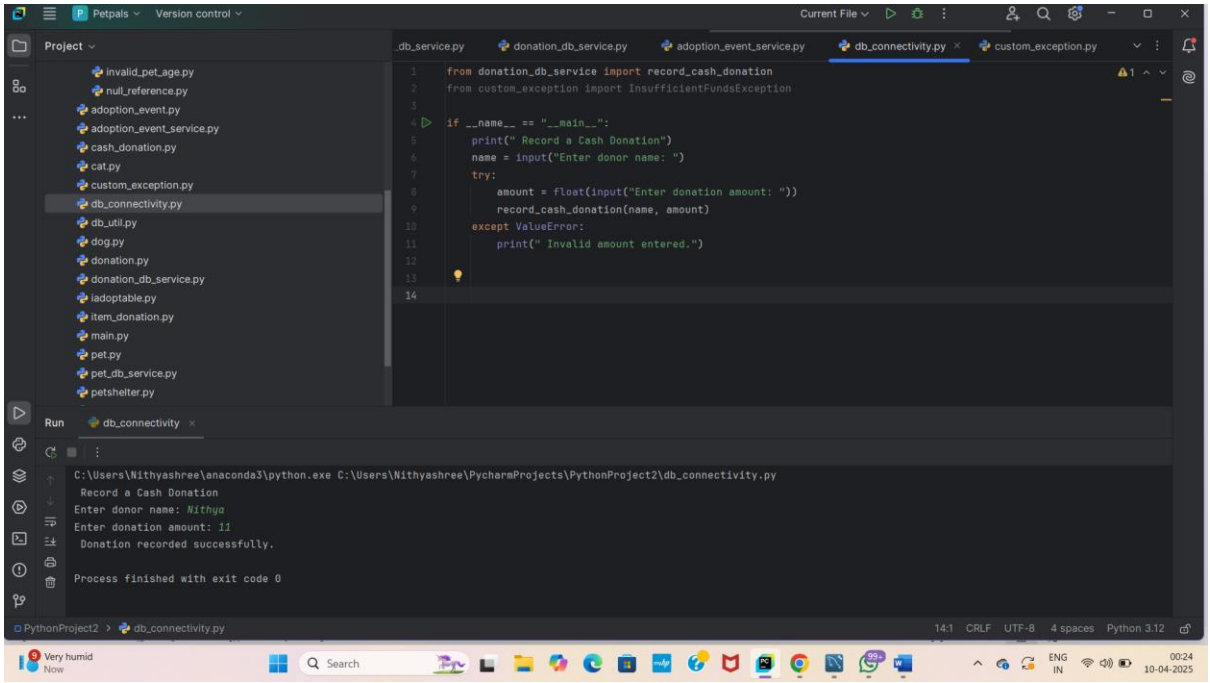
display_pets()



• 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.

donation_db_service.py:

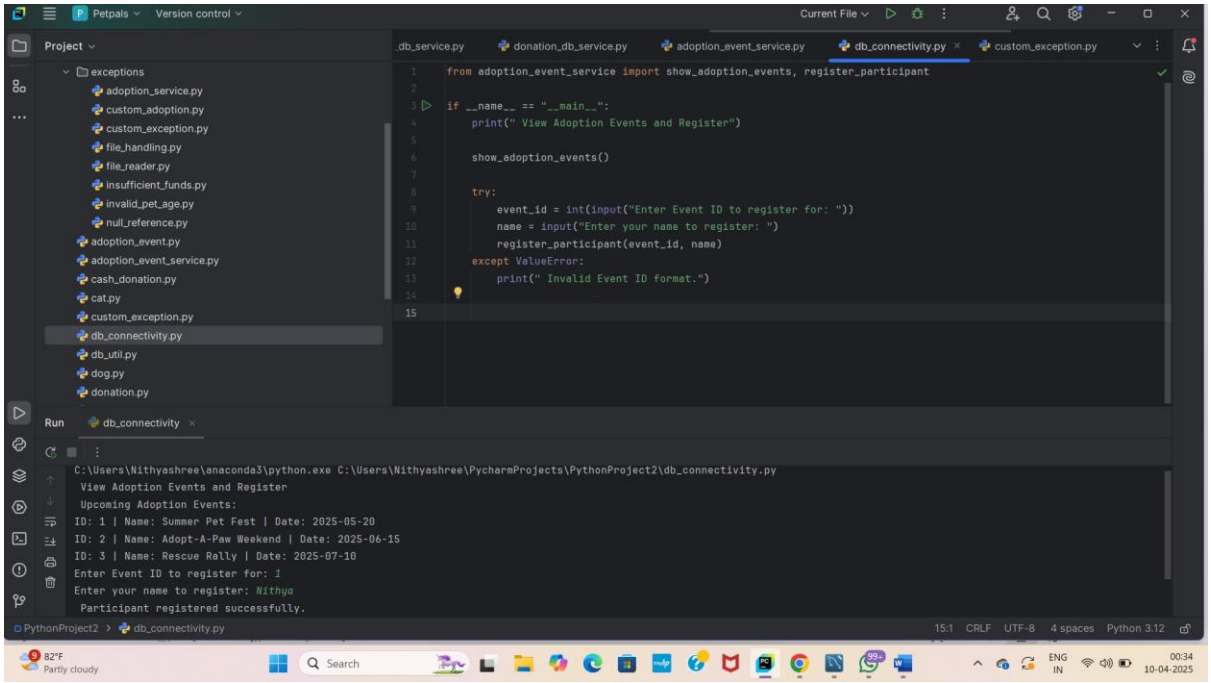
```
from donation_db_service import record_cash_donation
from custom_exception import InsufficientFundsException

if __name__ == "__main__":
    print(" Record a Cash Donation")
    name = input("Enter donor name: ")
    try:
        amount = float(input("Enter donation amount: "))
        record_cash_donation(name, amount)
    except ValueError:
        print(" Invalid amount entered.")
```



• 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.

adoption_event_service.py:

```
from adoption_event_service import show_adoption_events, register_participant

if __name__ == "__main__":
    print(" View Adoption Events and Register")

    show_adoption_events()

    try:
        event_id = int(input("Enter Event ID to register for: "))
        name = input("Enter your name to register: ")
        register_participant(event_id, name)
    except ValueError:
        print(" Invalid Event ID format.")
```


Petpals
Version control
Current File
Project
exceptions
adoption_service.py
custom_adoption.py
custom_exception.py
file_handling.py
file_reader.py
insufficient_funds.py
invalid_pet_age.py
null_reference.py
adoption_event.py
adoption_event_service.py
cash_donation.py
cat.py
custom_exception.py
db_connectivity.py
db_util.py
dog.py
donation.py
_db_service.py
donation_db_service.py
adoption_event_service.py
db_connectivity.py
custom_exception.py
from adoption_event_service import show_adoption_events, register_participant
if __name__ == "__main__":
print(" View Adoption Events and Register")
show_adoption_events()
try:
event_id = int(input("Enter Event ID to register for: "))
name = input("Enter your name to register: ")
register_participant(event_id, name)
except ValueError:
print(" Invalid Event ID format.")
Run
db_connectivity
C:\Users\Nithyashree\anaconda3\python.exe C:\Users\Nithyashree\PycharmProjects\PythonProject2\db_connectivity.py
View Adoption Events and Register
Upcoming Adoption Events:
ID: 1 | Name: Summer Pet Fest | Date: 2025-05-20
ID: 2 | Name: Adopt-A-Paw Weekend | Date: 2025-06-15
ID: 3 | Name: Rescue Rally | Date: 2025-07-10
Enter Event ID to register for: 1
Enter your name to register: Nithya
Participant registered successfully.
PythonProject2 > db_connectivity.py
15:1 CRLF UTF-8 4 spaces Python 3.12
82°F
Partly cloudy
Search
ENG
IN
00:34
10-04-2025