Name:- Dipti kumari

Coding Challenge:- PetPals, The Pet Adoption Platform

Github Repository:- https://github.com/dipti-kumari11/hexawarefilessubmission

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

Implement OOPs

Create SQL Schema from the pet and user class, use the class attributes for table column names. 1.Create and implement the mentioned class and the structure in your application. Pet Class: Attributes:

• Name (string): The name of the pet.

• Age (int): The age of the pet.

• Breed (string): The breed of the pet.

Methods:

• Constructor to initialize Name, Age, and Breed.

• Getters and setters for attributes.

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

**CODE:-**

package entity;

public class Pet {

private String name;

private int age;

private String breed;

public Pet(String name, int age, String breed) {

this.name = name;

this.age = age;

this.breed = breed;

}

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public int getAge() { return age; }

public void setAge(int age) { this.age = age; }

public String getBreed() { return breed; }

public void setBreed(String breed) { this.breed = breed; }

@Override

public String toString() {

return "Pet{name='" + name + "', age=" + age + ", breed='" + breed + "'}";

}

// Main method to test the Pet class

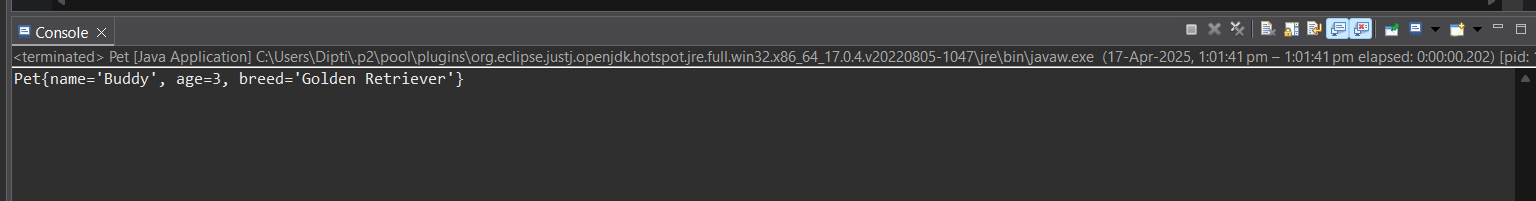
public static void main(String[] args) {

Pet myPet = new Pet("Buddy", 3, "Golden Retriever");

System.out.println(myPet); // Output: Pet{name='Buddy', age=3, breed='Golden Retriever'}

}

}

****

2. Dog Class (Inherits from Pet):

Additional Attributes:

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

Additional Methods:

• Constructor to initialize DogBreed.

• Getters and setters for DogBreed.

Cat Class (Inherits from Pet):

Additional Attributes:

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

Additional Methods:

• Constructor to initialize CatColor.

• Getters and setters for CatColor.

**CODE:-**

package entity;

public class Dog extends Pet {

private String dogBreed;

public Dog(String name, int age, String breed, String dogBreed) {

super(name, age, breed);

this.dogBreed = dogBreed;

}

public String getDogBreed() { return dogBreed; }

public void setDogBreed(String dogBreed) { this.dogBreed = dogBreed; }

*@Override*

public String toString() {

return super.toString() + ", DogBreed='" + dogBreed + "'";

}

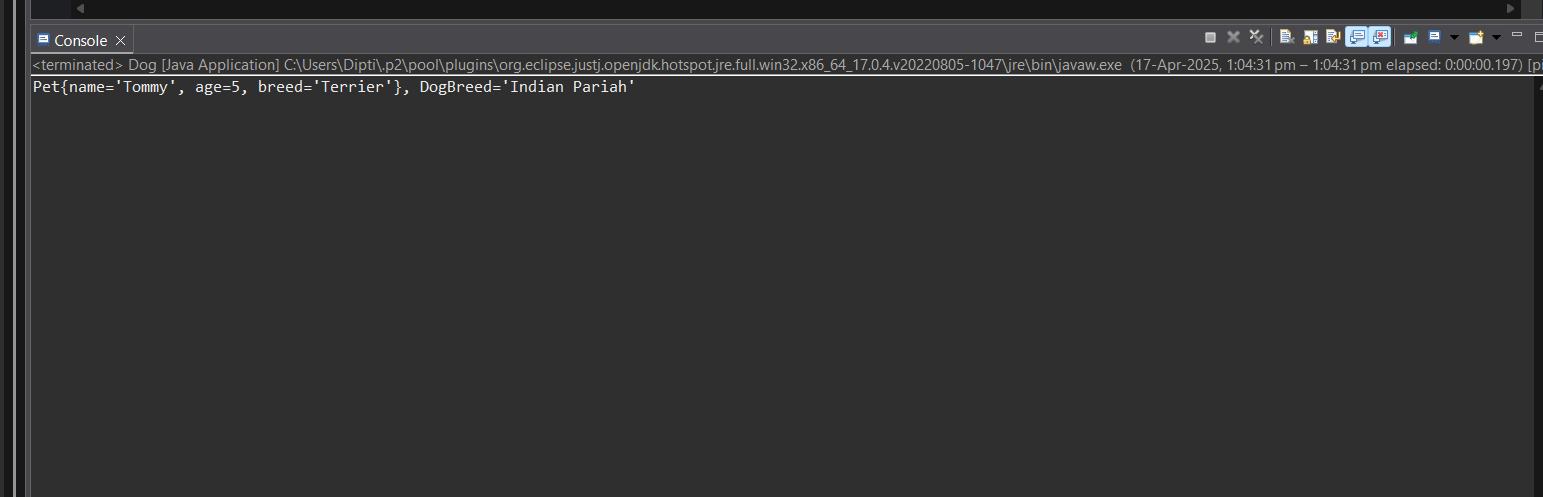
public static void main(String[] args) {

Dog myDog = new Dog("Tommy", 5, "Terrier", "Indian Pariah");

System.***out***.println(myDog); // Output: Pet{name='Tommy', age=5, breed='Terrier'}, DogBreed='Indian Pariah'

}

}



//CAT

package entity;

public class Cat extends Pet {

private String catColor;

public Cat(String name, int age, String breed, String catColor) {

super(name, age, breed);

this.catColor = catColor;

}

public String getCatColor() { return catColor; }

public void setCatColor(String catColor) { this.catColor = catColor; }

*@Override*

public String toString() {

return super.toString() + ", CatColor='" + catColor + "'";

}

public static void main(String[] args) {

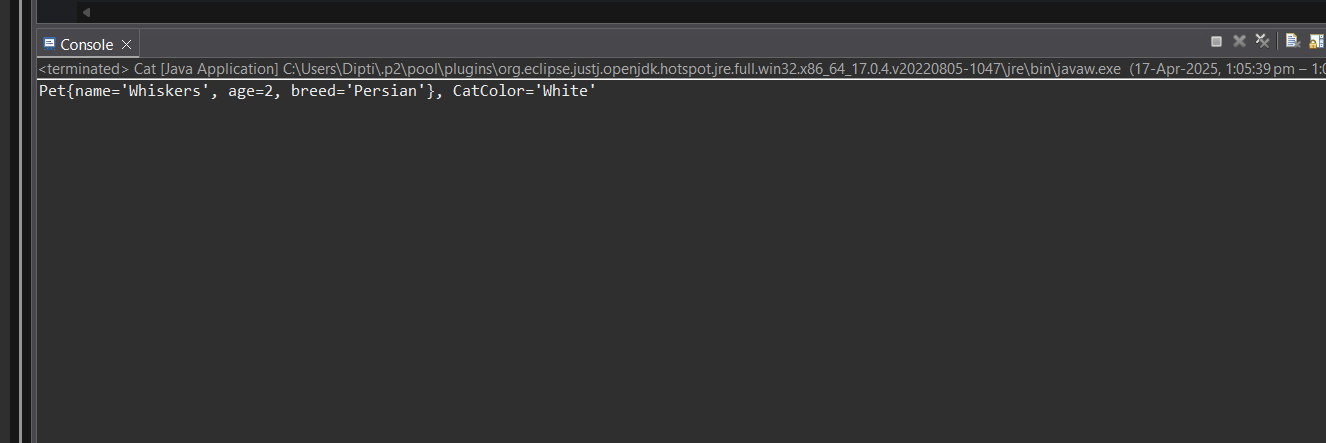
Cat myCat = new Cat("Whiskers", 2, "Persian", "White");

System.***out***.println(myCat);

// Output: Pet{name='Whiskers', age=2, breed='Persian'}, CatColor='White'

}

}



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.

**CODE:-**

package entity;

import java.util.ArrayList;

import java.util.List;

public class PetShelter {

private List<Pet> availablePets = new ArrayList<>();

public void addPet(Pet pet) {

availablePets.add(pet);

System.out.println("Pet added successfully.");

}

public void removePet(Pet pet) {

availablePets.remove(pet);

System.out.println("Pet removed successfully.");

}

public void listAvailablePets() {

if (availablePets.isEmpty()) {

System.out.println("No pets available in the shelter.");

} else {

for (Pet pet : availablePets) {

System.out.println(pet);

}

}

}

// Main method for testing

public static void main(String[] args) {

PetShelter shelter = new PetShelter();

Dog dog = new Dog("Rocky", 3, "Mixed", "Labrador");

Cat cat = new Cat("Luna", 2, "Persian", "Grey");

shelter.addPet(dog);

shelter.addPet(cat);

System.out.println("\nAvailable pets:");

shelter.listAvailablePets();

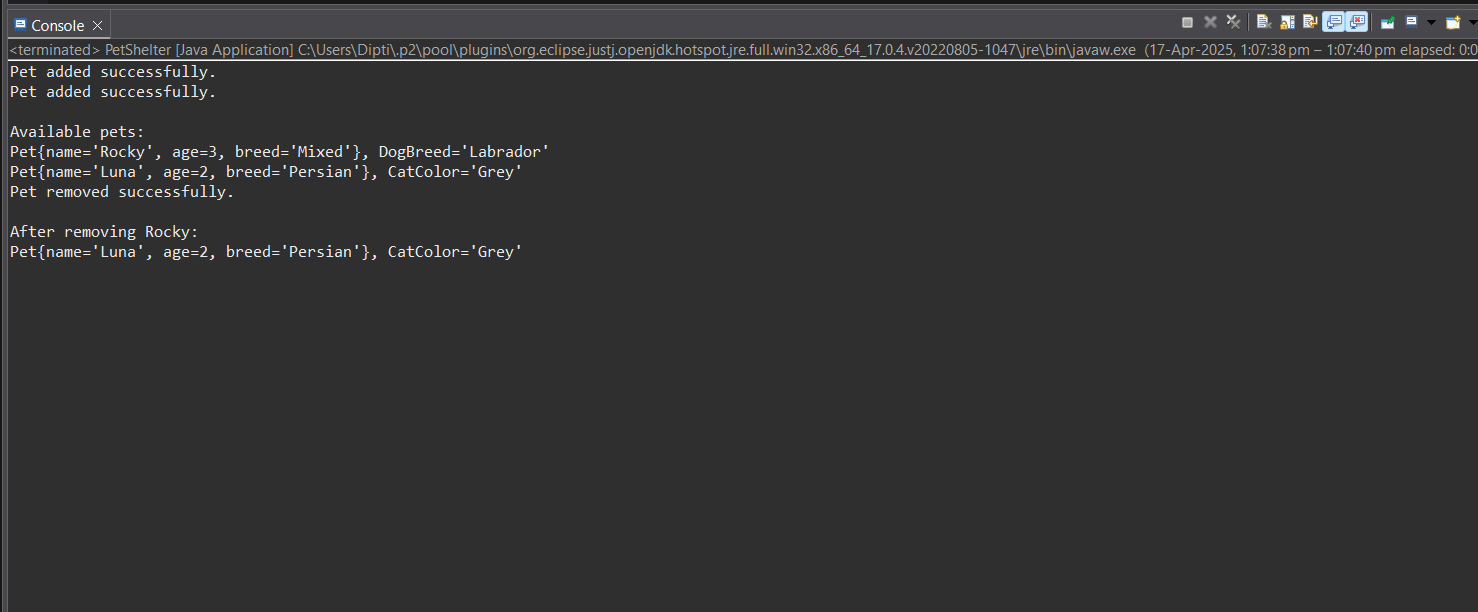
shelter.removePet(dog);

System.out.println("\nAfter removing Rocky:");

shelter.listAvailablePets();

}

}



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).

**CODE:-**

package Donation;

public abstract class Donation {

protected String donorName;

protected double amount;

public Donation(String donorName, double amount) {

this.donorName = donorName;

this.amount = amount;

}

public abstract void recordDonation();

public String getDonorName() { return donorName; }

public double getAmount() { return amount; }

}

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.

**CODE:-**

package Donation;

import java.time.LocalDate;

public class CashDonation extends Donation {

private LocalDate donationDate;

// Constructor to initialize donor details and donation date

public CashDonation(String donorName, double amount, LocalDate donationDate) {

super(donorName, amount);

this.donationDate = donationDate;

}

// Override to record the donation details

@Override

public void recordDonation() {

System.out.println("Cash donation of $" + amount + " recorded on " + donationDate + " by " + donorName);

}

// Getter for donation date

public LocalDate getDonationDate() {

return donationDate;

}

// Main method to test the functionality

public static void main(String[] args) {

// Creating a CashDonation instance

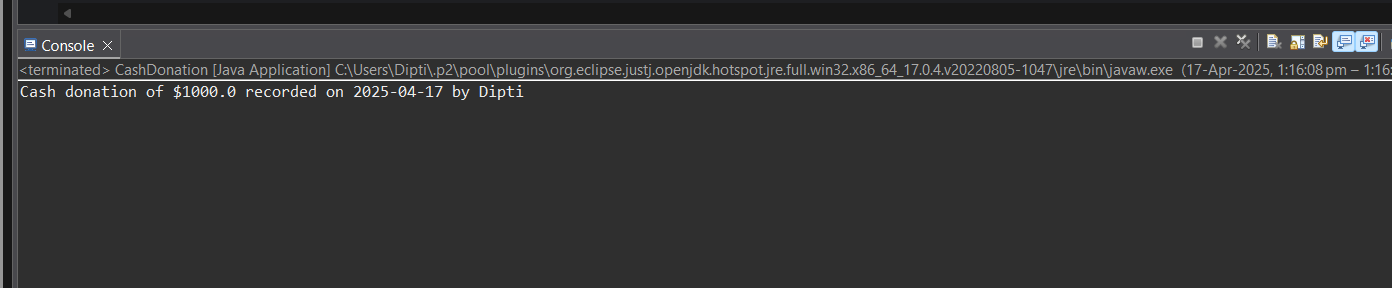
CashDonation donation = new CashDonation("Dipti", 1000, LocalDate.of(2025, 4, 17));

// Recording the donation

donation.recordDonation(); // Output: Cash donation of $1000.0 recorded on 2025-04-17 by Dipti

}

}



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.

**CODE:-**

package Donation;

public class ItemDonation extends Donation {

private String itemType;

private int quantity;

// Constructor to initialize donor details, amount, and item type

public ItemDonation(String donorName, double amount, String itemType, int quantity) {

super(donorName, amount); // Call parent constructor

this.itemType = itemType;

this.quantity = quantity;

}

// Override to record the donation details

@Override

public void recordDonation() {

System.out.println("Item donation: " + quantity + " units of " + itemType + " worth $" + amount

+ " donated by " + donorName);

}

// Getter for item type

public String getItemType() {

return itemType;

}

// Getter for quantity

public int getQuantity() {

return quantity;

}

// Main method for testing

public static void main(String[] args) {

// Creating an ItemDonation instance

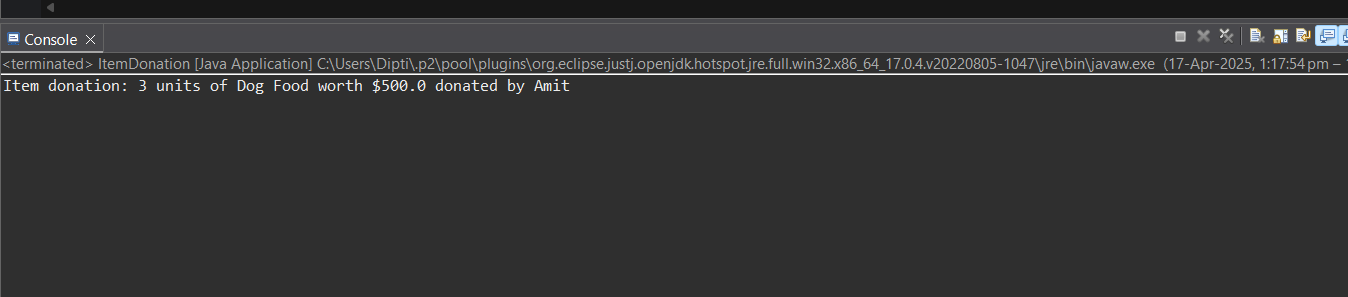
ItemDonation donation = new ItemDonation("Amit", 500, "Dog Food", 3);

// Recording the donation

donation.recordDonation(); // Output: Item donation: 3 units of Dog Food worth $500.0 donated by Amit

}

}



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.

**CODE:-**

// IAdoptable Interface/Abstract Class

package dao;

public interface IAdoptable {

void adopt();

}

// AdoptionEvent Class

package dao;

import java.util.ArrayList;

import java.util.List;

public class AdoptionEvent {

private List<IAdoptable> participants = new ArrayList<>();

public void registerParticipant(IAdoptable participant) {

participants.add(participant);

System.out.println("Participant registered.");

}

public void hostEvent() {

System.out.println("Hosting Adoption Event...");

for (IAdoptable participant : participants) {

participant.adopt();

}

}

}

// AdoptionEventTest

package dao;

import entity.Dog;

public class AdoptionEventTest {

public static void main(String[] args) {

// Create Adoption Event instance

AdoptionEvent event = new AdoptionEvent();

// Create Dog participants

Dog dog1 = new Dog("Buddy", 3, "Golden Retriever", "Lab Mix");

Dog dog2 = new Dog("Max", 4, "Labrador", "Chihuahua Mix");

// Register participants to the adoption event

event.registerParticipant(dog1);

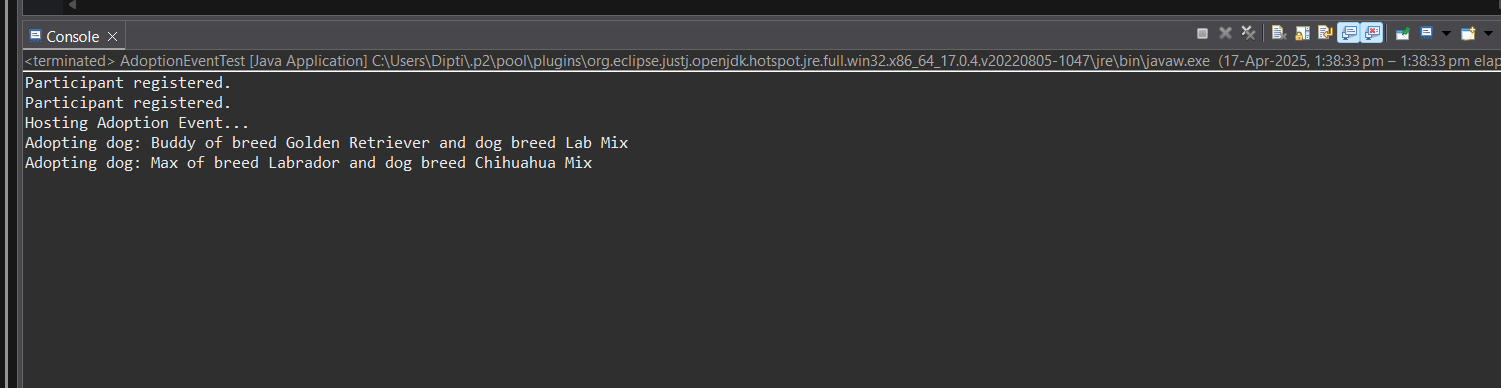
event.registerParticipant(dog2);

// Host the adoption event and perform adoption for each participant

event.hostEvent();

}

}



6.Exceptions handling

Create and implement the following exceptions in your application.

• Invalid Pet Age Handling:

o In the Pet Adoption Platform, when adding a new pet to a shelter, the age of the pet should

be a positive integer. Write a program that prompts the user to input the age of a pet.

Implement exception handling to ensure that the input is a positive integer. If the input is

not valid, catch the exception and display an error message. If the input is valid, add the pet

to the shelter.

• Null Reference Exception Handling:

o In the Pet Adoption Platform, when displaying the list of available pets in a shelter, it's

important to handle situations where a pet's properties (e.g., Name, Age) might be null.

Implement exception handling to catch null reference exceptions when accessing properties

of pets in the shelter and display a message indicating that the information is missing.

• Insufficient Funds Exception:

o Suppose the Pet Adoption Platform allows users to make cash donations to shelters. Write a

program that prompts the user to enter the donation amount. Implement exception

handling to catch situations where the donation amount is less than a minimum allowed

amount (e.g., $10). If the donation amount is insufficient, catch the exception and display an

error message. Otherwise, process the donation.

• File Handling Exception:

o In the Pet Adoption Platform, there might be scenarios where the program needs to read

data from a file (e.g., a list of pets in a shelter). Write a program that attempts to read data

from a file. Implement exception handling to catch any file-related exceptions (e.g.,

FileNotFoundException) and display an error message if the file is not found or cannot be

read.

• Custom Exception for Adoption Errors:

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

**CODE:-**

// InvalidPetAgeDemo

package entity;

import java.util.InputMismatchException;

import java.util.Scanner;

public class InvalidPetAgeDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

try {

System.out.print("Enter pet age: ");

int age = sc.nextInt();

if (age <= 0) {

throw new IllegalArgumentException("Pet age must be a positive integer.");

}

System.out.println("Pet added with age: " + age);

} catch (InputMismatchException e) {

System.out.println("Invalid input! Please enter an integer.");

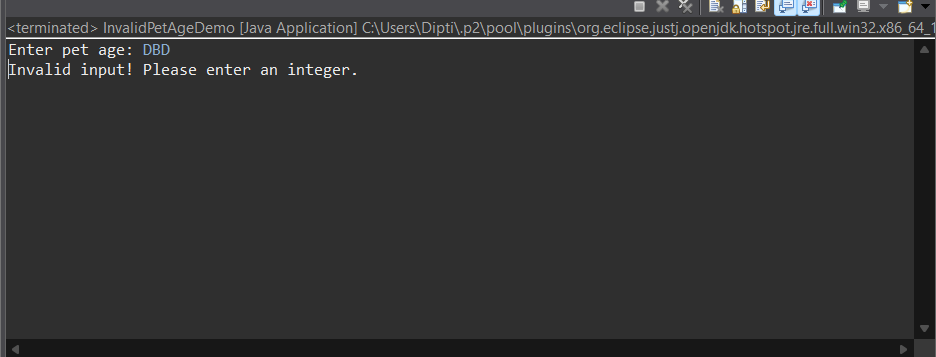
} catch (IllegalArgumentException e) {

System.out.println("Error: " + e.getMessage());

}

}

}



// Null Reference Exception Handling

package entity;

public class NullReferenceDemo {

static class Pet {

String name;

Integer age;

public Pet(String name, Integer age) {

this.name = name;

this.age = age;

}

}

//

public static void main(String[] args) {

Pet[] pets = {

new Pet("Buddy", 2),

new Pet(null, 4),

new Pet("Charlie", null)

};

for (Pet pet : pets) {

try {

System.***out***.println("Name: " + pet.name.toUpperCase() + ", Age: " + pet.age);

} catch (NullPointerException e) {

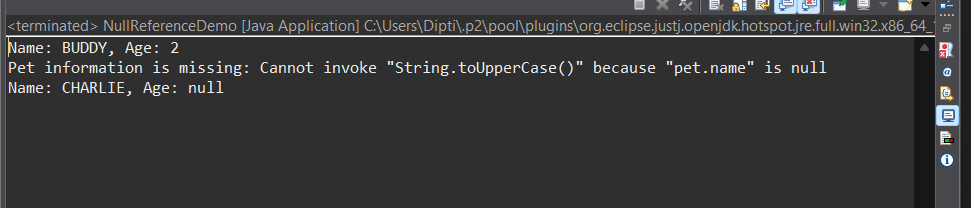
System.***out***.println("Pet information is missing: " + e.getMessage());

}

}

}

}



// Insufficient Funds Exception

package entity;

import java.util.Scanner;

// Custom Exception

class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

public class DonationDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter donation amount: ");

double amount = sc.nextDouble();

try {

if (amount < 10) {

throw new InsufficientFundsException("Donation must be at least $10.");

}

System.***out***.println("Thank you for your donation of $" + amount);

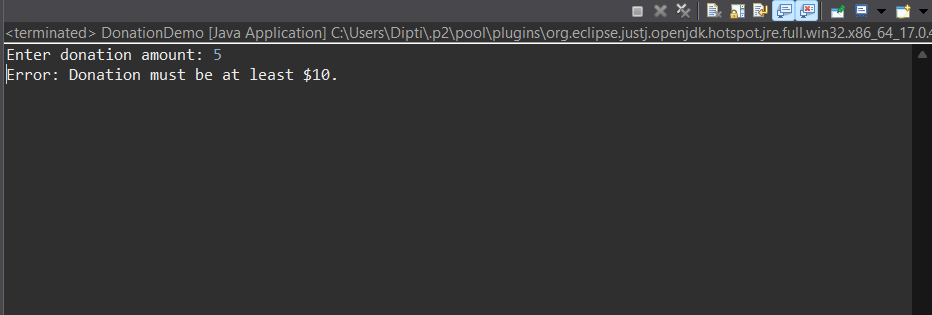
} catch (InsufficientFundsException e) {

System.***out***.println("Error: " + e.getMessage());

}

}

}



// File Handling Exception

package entity;

import java.io.\*;

public class FileReadDemo {

public static void main(String[] args) {

try {

FileReader fr = new FileReader("pets.txt");

BufferedReader br = new BufferedReader(fr);

String line;

while ((line = br.readLine()) != null) {

System.***out***.println("Pet: " + line);

}

br.close();

} catch (FileNotFoundException e) {

System.***out***.println("Error: File not found.");

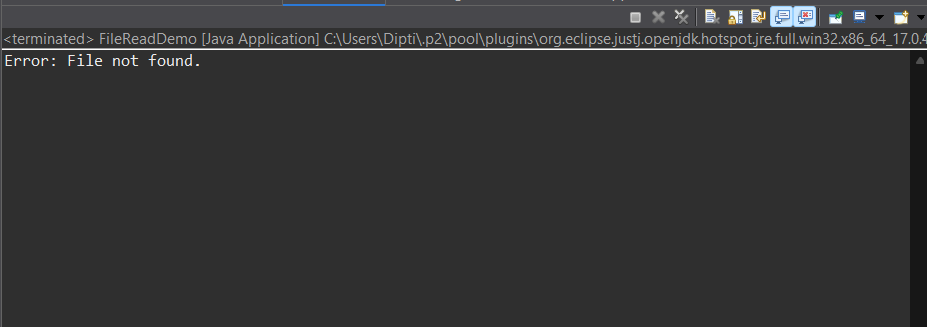
} catch (IOException e) {

System.***out***.println("Error reading the file.");

}

}

}



// Custom Exception for Adoption Errors

package entity;

// Custom exception for adoption-related errors

class AdoptionException extends Exception {

public AdoptionException(String message) {

super(message);

}

}

// Pet class with adoption logic

public class AdoptionDemo {

static class Pet {

String name;

boolean available;

Pet(String name, boolean available) {

this.name = name;

this.available = available;

}

public void adopt() throws AdoptionException {

if (!available || name == null) {

throw new AdoptionException("Adoption failed: Pet is not available or info missing.");

}

System.***out***.println("Adoption successful for: " + name);

}

}

public static void main(String[] args) {

Pet pet1 = new Pet("Bella", false); // Not available

Pet pet2 = new Pet(null, true); // Missing name

Pet pet3 = new Pet("Rocky", true); // Valid pet

try {

pet1.adopt();

} catch (AdoptionException e) {

System.***out***.println("Error1: " + e.getMessage());

}

try {

pet2.adopt();

} catch (AdoptionException e) {

System.***out***.println("Error2: " + e.getMessage());

}

try {

pet3.adopt();

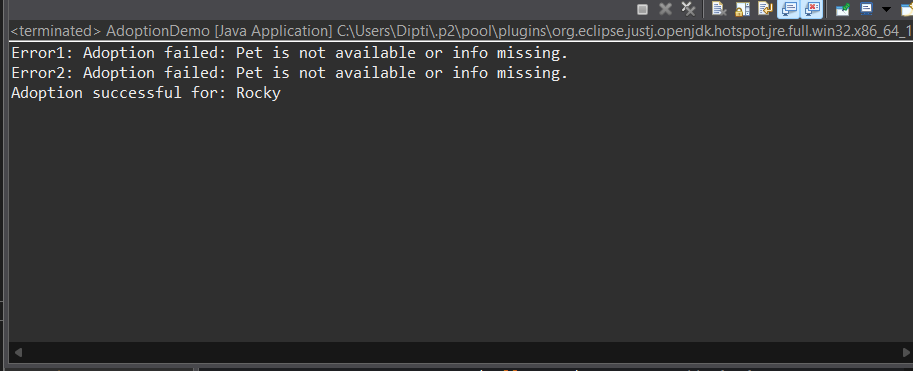
} catch (AdoptionException e) {

System.***out***.println("Error3: " + e.getMessage());

}

}

}



7.Database Connectivity

Create and implement the following tasks in your application.

• Displaying Pet Listings:

o Develop a program that connects to the database and retrieves a list of available pets

from the "pets" table. Display this list to the user. Ensure that the program handles

database connectivity exceptions gracefully, including cases where the database is

unreachable.

• Donation Recording:

o Create a program that records cash donations made by donors. Allow the user to input

donor information and the donation amount and insert this data into the "donations"

table in the database. Handle exceptions related to database operations, such as

database errors or invalid inputs.

• Adoption Event Management:

o Build a program that connects to the database and retrieves information about

upcoming adoption events from the "adoption\_events" table. Allow the user to register

for an event by adding their details to the "participants" table. Ensure that the program

handles database connectivity and insertion exceptions properly.

**CODE:-**

// Displaying Pet Listings

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class SelectTest {

public static void main(String[] args) throws Exception

{

Class.forName("com.mysql.cj.jdbc.Driver");

// variables

final String url = "jdbc:mysql:///petpals";

final String user = "root";

final String password = "root";

// establish the connection

Connection con = DriverManager.getConnection(url, user, password);

// create JDBC statement object

Statement st = con.createStatement();

// prepare SQL query

String query = "SELECT PetID, NAME, Age,Breed,Type,AvailableForAdoption FROM pets";

// send and execute SQL query in Database software

ResultSet rs = st.executeQuery(query);

// process the ResultSet object

boolean flag = false;

while (rs.next()) {

flag = true;

System.out.println(rs.getInt(1) + " | " + rs.getString(2) +

" | " + rs.getString(3)+ " | "+rs.getString(4)+" | "+rs.getString(5)+" | "+rs.getString(6));

}

if (flag == true) {

System.out.println("\nRecords retrieved and displayed");

} else {

System.out.println("Record not found");

}

// close JDBC objects

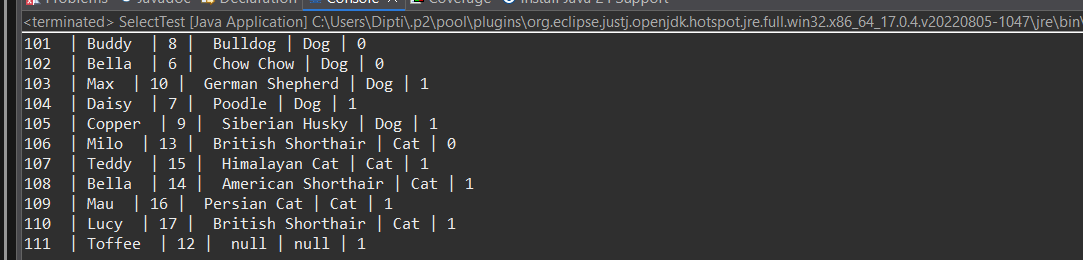
rs.close();

st.close();

con.close();

}

}



// Donation Recording

package util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class DonationRecords {

public static void main(String[] args) throws Exception

{

Class.forName("com.mysql.cj.jdbc.Driver");

// variables

final String url = "jdbc:mysql:///petpals";

final String user = "root";

final String password = "root";

// establish the connection

Connection con = DriverManager.getConnection(url, user, password);

// create JDBC statement object

Statement st = con.createStatement();

// prepare SQL query

String query = "Insert into donations(DonationID, DonorName, DonationType,DonationAmount,DonationItem,DonationDate) Values (12, 'Krishi', 'Cash', 120.00, NULL, '2025-03-01 10:37:00')";

// send and execute SQL query in Database software

//ResultSet rs = st.executeQuery(query);

int n=st.executeUpdate(query);

if (n==1)

System.out.println("\nRecords inserted");

else

System.out.println("Record not inserted") ;

// process the ResultSet object

/\* boolean flag = false;

while (rs.next()) {

flag = true;

System.out.println(rs.getInt(1) + " " + rs.getString(2) +

" " + rs.getString(3));

}

if (flag == true) {

System.out.println("\nRecords retrieved and displayed");

} else {

System.out.println("Record not found");

}\*/

// close JDBC objects

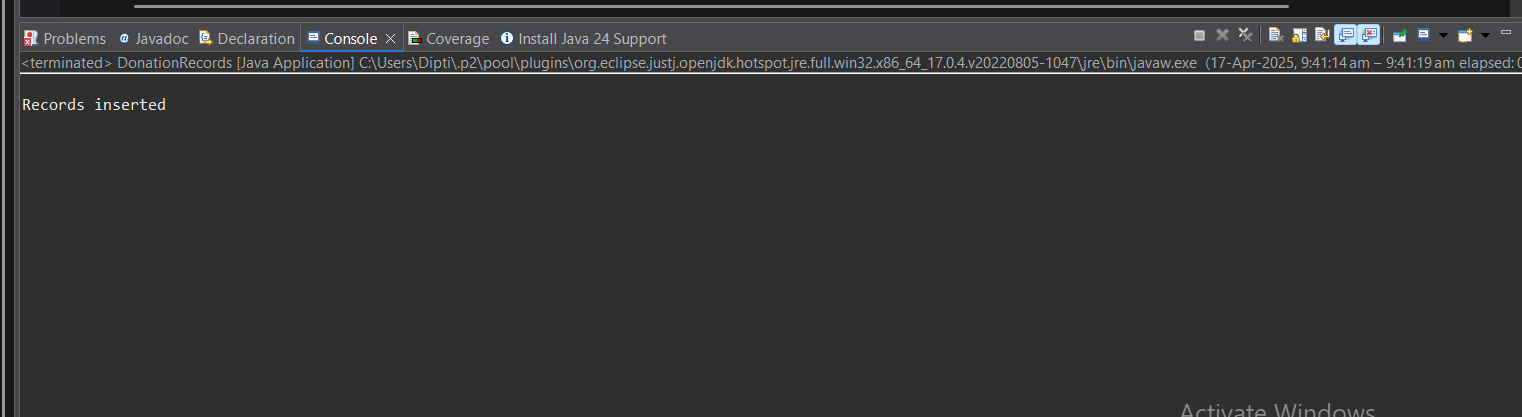
// rs.close();

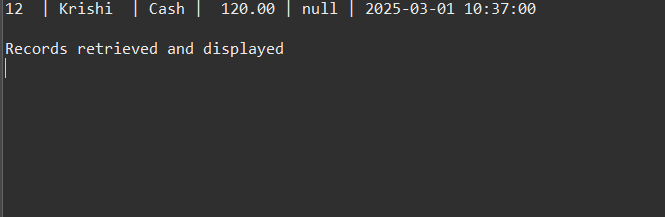
st.close();

con.close();

}

}





// Adoption Event Management

package util;

import java.sql.\*;

import java.util.Scanner;

public class AdoptionEventManager {

public static void main(String[] args) {

// 🔐 Replace with your actual DB details

String url = "jdbc:mysql://localhost:3306/petpals";

String user = "root";

String password = "root";

Scanner scanner = new Scanner(System.***in***);

try (Connection con = DriverManager.*getConnection*(url, user, password);

Statement stmt = con.createStatement()) {

ResultSet rs = stmt.executeQuery("SELECT \* FROM adoptionevents");

System.***out***.println("📋 Upcoming Adoption Events:");

while (rs.next()) {

System.***out***.println("Event ID: " + rs.getInt("EventID") +

", Name: " + rs.getString("EventName") +

", Date: " + rs.getDate("EventDate"));

}

System.***out***.print("Enter Event ID to register: ");

int eventId = Integer.*parseInt*(scanner.nextLine());

System.***out***.print("Enter your name: ");

String name = scanner.nextLine();

String sql = "INSERT INTO participants (ParticipantName, EventID) VALUES (?, ?)";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setString(1, name);

pstmt.setInt(2, eventId);

pstmt.executeUpdate();

System.***out***.println("✅ Registered successfully for the event!");

}

} catch (SQLException e) {

System.***out***.println("❌ DB Error: " + e.getMessage());

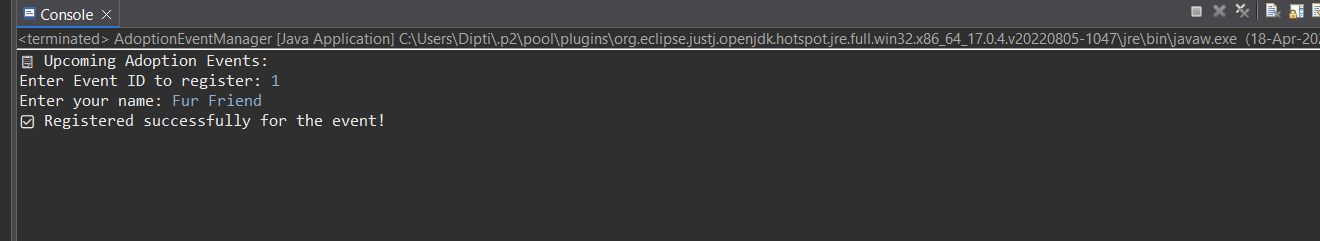
} finally {

scanner.close();

}

}

}



DROP TABLE IF EXISTS participants;

DROP TABLE IF EXISTS donations;

DROP TABLE IF EXISTS adoptionevents;

DROP TABLE IF EXISTS pets;

DROP TABLE IF EXISTS adoptions;

DROP TABLE IF EXISTS shelters;

create table Pets(

PetID int primary key,

Name Varchar(20),

Age int,

Breed Varchar(30),

Type Varchar(20),

AvailableForAdoption BIT DEFAULT 1 CHECK (AvailableForAdoption IN (0, 1))

);

create table Shelters(

ShelterID int primary key,

Name Varchar(20),

Location Varchar(50));

create table Donations(

DonationID int primary key,

DonorName Varchar(30),

DonationType Varchar(20),

DonationAmount DECIMAL(10, 2),

DonationItem Varchar(30),

DonationDate datetime

);

create table AdoptionEvents(

EventID int primary key,

EventName Varchar(30),

EventDate datetime,

Location Varchar(50));

Create table Participants(

ParticipantID int AUTO\_INCREMENT primary key,

ParticipantName Varchar(20),

ParticipantType Varchar(20),

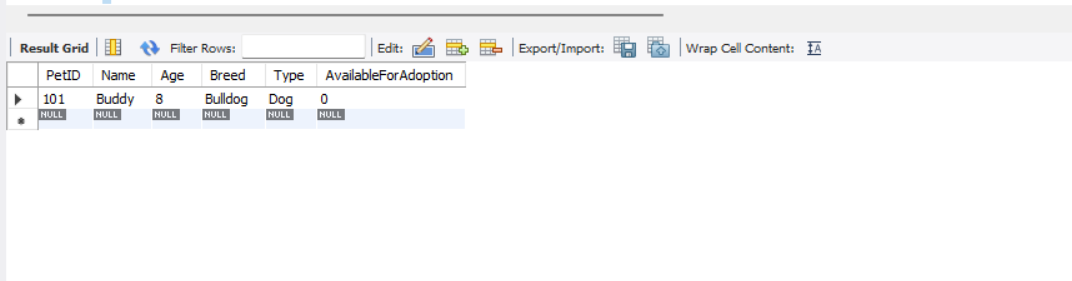
EventID INT,

FOREIGN KEY (EventID) REFERENCES AdoptionEvents(EventID)

);

insert into pets values(101, 'Buddy',8,'Bulldog','Dog',0);

select \* from pets;



INSERT INTO AdoptionEvents VALUES (1011, 'Spring Adopt Fest', '2025-03-21 10:00:00', 'Central Park');

INSERT INTO AdoptionEvents VALUES (2021, 'Paws & Hearts Day', '2025-02-14 11:00:00', 'City Hall Grounds');

select \* from AdoptionEvents;

