Practical Project Part 3

Full Name: Meet Maheta

Course Name: CST8333

Assignment Title: Programming Language Research Project - Practical Project Part 3

Submission Date: July 21, 2024

Evidence of Learning

**1. Inheritance and Polymorphism**

* Polymorphism allows objects of different classes to be treated as objects of a common super class. It is implemented in this project by creating a superclass RecordBase and a subclass TrafficRecord that overrides a method from the superclass.

A screen shot of a computer

Description automatically generated

**Sub-Class Example:**

A screen shot of a computer program

Description automatically generated

**2. Database Connectivity:**

* Description: This project uses MySQL for database connectivity. The traffic\_manager class manages connections to the database and performs CRUD operations.

A screen shot of a computer program

Description automatically generated

**Explanation of Key Lines:**

* Line 24: Initializes the database connection when creating a TrafficManager instance.
* Line 25: Loads data from the database if the connection is successful.
* Line 32: Defines the connect\_to\_db method which establishes a connection to the MySQL database.
* Line 53: Creates a cursor object to execute SQL commands.
* Line 75: Commits the transaction to the database, ensuring the table creation is saved.

Program Changes:

**Description**: The project is structured using a layered architecture. The RecordBase and Record classes demonstrate inheritance and polymorphism, while the TrafficManager class handles database connectivity.

**Outline**:

* **Polymorphism**: Implemented in RecordBase and Record classes.
* **Database Connectivity**: Managed by TrafficManager class.

**Explanation**:

* **Polymorphism**: The display method in Record overrides the method in RecordBase.
* **Database Connectivity**: Methods for connecting to the database and performing CRUD operations are in TrafficManager.

Program Demonstration via Screen Shots

### **Introduction:**

This section includes screenshots demonstrating various functionalities of the traffic data management application, showcasing both database connectivity and polymorphism.

1. **Initial Data Load** **& Database schema:**

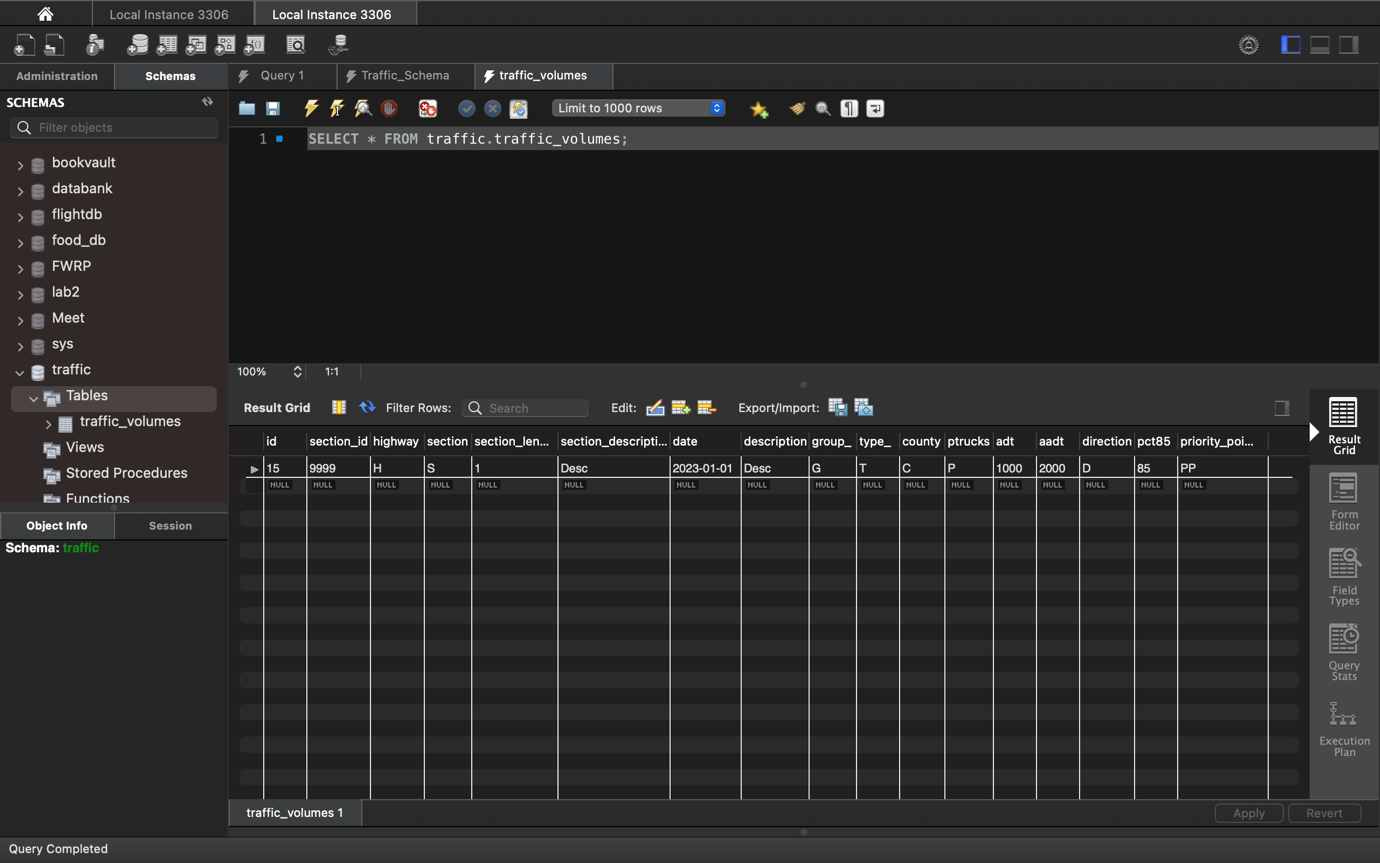
The screenshot illustrates the initial loading of traffic data from the MySQL database into the application. It confirms that the data is successfully fetched and displayed, with my full name, Meet Maheta, visible to ensure authenticity.

Created Table and database for the program to store data and perform CRUD:

A screenshot of a computer

Description automatically generated

This is the screenshot of loaded data using test program just to show the added data in the table:



The screenshot below shows the presentation of already added data from the database:

A screenshot of a computer

Description automatically generated

1. **Add Record:**

 This screenshot captures the process of adding a new record to the database. It shows the form used to input new data and the subsequent confirmation message, indicating that the record has been successfully added. My full name is visible in the screenshot for verification.

A screenshot of a computer

Description automatically generated

This screenshot below illustrates that the data has been stored successfully in the database:

A screenshot of a computer

Description automatically generated

1. **Edit Record:**

This screenshot demonstrates editing an existing record in the database. It displays the form used to update the data and the confirmation message that appears upon successful modification of the record. My full name is clearly visible to maintain authenticity.

A screenshot of a computer

Description automatically generated

This screenshot below illustrates that the data has been edited successfully in the database at index 20:

A screenshot of a computer

Description automatically generated

1. **Delete Record:**

The screenshot shows the deletion of a record from the database. It includes the confirmation message verifying that the selected record has been successfully removed. As always, my full name is included in the screenshot to verify the authenticity of the work.

A screenshot of a computer

Description automatically generated

This screenshot below illustrates that the data has been deleted successfully in the database at index 21:

A screenshot of a computer

Description automatically generated

1. **Demonstrate Polymorphism:**

This screenshot exemplifies the use of polymorphism in the application. It shows the output of polymorphic method calls, where different subclasses of record\_base display their specific output formats. My full name is displayed to ensure the authenticity of the demonstration.

A screenshot of a computer

Description automatically generated

**These screenshots collectively demonstrate the functionality and advanced features of the application, including robust database connectivity for CRUD operations and the implementation of polymorphism to manage traffic records efficiently. Each screenshot includes my full name, Meet Maheta, to verify the authenticity and integrity of the project.**

Unit Testing Demonstration via Screen Shots

**Introduction:**

This section includes a screenshot demonstrating the unit testing of the edit\_record method, ensuring the functionality of the application is thoroughly tested and validated.

**Running Unit Test:** The screenshot showcases the execution of the unit test for the edit\_record method within the TrafficManager class. It confirms that the method correctly edits an existing record in the database, as verified by the successful test run. The screenshot includes my full name, Meet Maheta, ensuring the authenticity and integrity of the test results.

**Running Unit Tests from Console**

**A screen shot of a computer

Description automatically generated**

As you can see, we have deleted this record before to perform delete record step and there was only one record left in the table, but after running test class, we have the same record again that indicates the test for edit existing record work perfectly.

**A screenshot of a computer

Description automatically generated**

Source Code Commenting Example

**traffic\_manager.py**

# traffic\_manager.py

# Author: Meet Maheta

import mysql.connector

from mysql.connector import Error

from record import TrafficRecord

from special\_traffic\_record import SpecialTrafficRecord

class TrafficManager:

"""

The TrafficManager class manages traffic records using MySQL database.

"""

def \_\_init\_\_(self, host, database, user, password):

"""

Initialize the TrafficManager with database connection details.

Parameters:

-----------

host : str

The hostname of the MySQL database.

database : str

The name of the MySQL database.

user : str

The username for the MySQL database.

password : str

The password for the MySQL database.

"""

self.host = host

self.database = database

self.user = user

self.password = password

self.conn = self.connect\_to\_db() # Connect to the database

self.records = self.load\_data() if self.conn else [] # Load data if connection is successful

def connect\_to\_db(self):

"""

Establish a connection to the MySQL database.

Returns:

--------

conn : mysql.connector.connection.MySQLConnection or None

The MySQL connection object if successful, None otherwise.

"""

try:

conn = mysql.connector.connect(

host=self.host,

database=self.database,

user=self.user,

password=self.password

)

if conn.is\_connected():

print("Connected to MySQL database")

return conn

except Error as e:

print(f"Error: {e}")

return None

def create\_table(self):

"""

Create the traffic\_volumes table if it does not exist.

"""

if not self.conn:

print("Failed to connect to the database.")

return

try:

cursor = self.conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS traffic\_volumes (

id INT AUTO\_INCREMENT PRIMARY KEY,

section\_id VARCHAR(255),

highway VARCHAR(255),

section VARCHAR(255),

section\_length FLOAT,

section\_description VARCHAR(255),

date DATE,

description VARCHAR(255),

group\_ VARCHAR(255),

type\_ VARCHAR(255),

county VARCHAR(255),

ptrucks VARCHAR(255),

adt FLOAT,

aadt FLOAT,

direction VARCHAR(255),

pct85 VARCHAR(255),

priority\_points VARCHAR(255)

)

''')

self.conn.commit()

except Error as e:

print(f"Database error: {e}")

def load\_data(self):

"""

Load data from the traffic\_volumes table into the records list.

Returns:

--------

records : list of TrafficRecord

A list of TrafficRecord objects loaded from the database.

"""

if not self.conn:

print("Failed to connect to the database.")

return []

try:

records = []

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM traffic\_volumes")

rows = cursor.fetchall()

for row in rows:

record = TrafficRecord(\*row[1:]) # Exclude the auto-incremented ID

records.append(record)

return records

except Error as e:

print(f"Database error: {e}")

return []

def save\_data(self):

"""

Save the records list to the traffic\_volumes table.

"""

if not self.conn:

print("Failed to connect to the database.")

return

try:

cursor = self.conn.cursor()

cursor.execute("DELETE FROM traffic\_volumes") # Clear existing records

for record in self.records:

cursor.execute('''

INSERT INTO traffic\_volumes (

section\_id, highway, section, section\_length, section\_description,

date, description, group\_, type\_, county, ptrucks, adt, aadt,

direction, pct85, priority\_points

) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

''', (

record.section\_id, record.highway, record.section, record.section\_length,

record.section\_description, record.date, record.description, record.group,

record.type\_, record.county, record.ptrucks, record.adt, record.aadt,

record.direction, record.pct85, record.priority\_points

))

self.conn.commit()

except Error as e:

print(f"Database error: {e}")

def add\_record(self, record\_data):

"""

Add a new record to the records list and the database.

Parameters:

-----------

record\_data : dict

A dictionary containing the data for the new record.

"""

if not self.conn:

print("Failed to connect to the database.")

return

record = TrafficRecord(\*\*record\_data)

self.records.append(record)

try:

cursor = self.conn.cursor()

cursor.execute('''

INSERT INTO traffic\_volumes (

section\_id, highway, section, section\_length, section\_description,

date, description, group\_, type\_, county, ptrucks, adt, aadt,

direction, pct85, priority\_points

) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

''', (

record.section\_id, record.highway, record.section, record.section\_length,

record.section\_description, record.date, record.description, record.group,

record.type\_, record.county, record.ptrucks, record.adt, record.aadt,

record.direction, record.pct85, record.priority\_points

))

self.conn.commit()

except Error as e:

print(f"Database error: {e}")

def edit\_record(self, record\_id, new\_data):

"""

Edit an existing record in the database.

Parameters:

-----------

record\_id : int

The ID of the record to edit.

new\_data : dict

A dictionary containing the updated data for the record.

Returns:

--------

bool

True if the record was successfully edited, False otherwise.

"""

if not self.conn:

print("Failed to connect to the database.")

return False

try:

cursor = self.conn.cursor()

cursor.execute('''

UPDATE traffic\_volumes SET

section\_id=%s, highway=%s, section=%s, section\_length=%s,

section\_description=%s, date=%s, description=%s, group\_=%s, type\_=%s,

county=%s, ptrucks=%s, adt=%s, aadt=%s, direction=%s, pct85=%s, priority\_points=%s

WHERE id=%s

''', (

new\_data['section\_id'], new\_data['highway'], new\_data['section'], new\_data['section\_length'],

new\_data['section\_description'], new\_data['date'], new\_data['description'], new\_data['group'],

new\_data['type\_'], new\_data['county'], new\_data['ptrucks'], new\_data['adt'], new\_data['aadt'],

new\_data['direction'], new\_data['pct85'], new\_data['priority\_points'], record\_id

))

self.conn.commit()

return cursor.rowcount > 0 # Return True if a record was updated

except Error as e:

print(f"Database error: {e}")

return False

def delete\_record(self, record\_id):

"""

Delete a record from the database.

Parameters:

-----------

record\_id : int

The ID of the record to delete.

Returns:

--------

bool

True if the record was successfully deleted, False otherwise.

"""

if not self.conn:

print("Failed to connect to the database.")

return False

try:

cursor = self.conn.cursor()

cursor.execute("DELETE FROM traffic\_volumes WHERE id=%s", (record\_id,))

self.conn.commit()

return cursor.rowcount > 0 # Return True if a record was deleted

except Error as e:

print(f"Database error: {e}")

return False

def get\_record(self, record\_id):

"""

Retrieve a record from the database by ID.

Parameters:

-----------

record\_id : int

The ID of the record to retrieve.

Returns:

--------

TrafficRecord or None

The TrafficRecord object if found, None otherwise.

"""

if not self.conn:

print("Failed to connect to the database.")

return None

try:

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM traffic\_volumes WHERE id=%s", (record\_id,))

row = cursor.fetchone()

if row:

return TrafficRecord(\*row[1:]) # Exclude the auto-incremented ID

return None

except Error as e:

print(f"Database error: {e}")

return None

def reload\_data(self):

"""

Reload the records list from the database.

"""

self.records = self.load\_data()

def demonstrate\_polymorphism(self):

"""

Demonstrate polymorphism with TrafficRecord and SpecialTrafficRecord.

"""

sample\_records = [

TrafficRecord("1", "Highway1", "Section1", "10", "Description1", "2023-01-01", "Desc1", "Group1", "Type1", "County1", "10", "1000", "2000", "North", "85", "10"),

SpecialTrafficRecord("2", "Highway2", "Section2", "20", "Description2", "2023-01-02", "Desc2", "Group2", "Type2", "County2", "20", "2000", "3000", "South", "90", "20")

]

for record in sample\_records:

print(record.display())

**business.py**

# business.py

# Author: Meet Maheta

import csv

from record import TrafficRecord

from special\_traffic\_record import SpecialTrafficRecord

class TrafficManager:

"""

The TrafficManager class manages a collection of RecordBase objects.

It provides methods to load, save, add, edit, delete, and retrieve records.

"""

def \_\_init\_\_(self, filename):

"""

Initialize the TrafficManager with the given filename.

:param filename: The path to the CSV file containing traffic data.

"""

self.filename = filename

self.records = self.load\_data(filename)

def load\_data(self, filename):

"""

Load data from a CSV file and create TrafficRecord objects.

:param filename: The path to the CSV file containing traffic data.

:return: A list of TrafficRecord objects.

"""

try:

records = []

with open(filename, newline='') as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

record = TrafficRecord(

row['SECTION ID'], row['HIGHWAY'], row['SECTION'], row['SECTION LENGTH'],

row['SECTION DESCRIPTION'], row['Date'], row['DESCRIPTION'], row['GROUP'],

row['TYPE'], row['COUNTY'], row['PTRUCKS'], row['ADT'], row['AADT'],

row['DIRECTION'], row['85PCT'], row['PRIORITY\_POINTS']

)

records.append(record)

return records

except FileNotFoundError:

print("File not found.")

return []

def save\_data(self):

"""

Save the current list of TrafficRecord objects to a CSV file.

"""

with open(self.filename, mode='w', newline='') as csvfile:

fieldnames = ['SECTION ID', 'HIGHWAY', 'SECTION', 'SECTION LENGTH', 'SECTION DESCRIPTION',

'Date', 'DESCRIPTION', 'GROUP', 'TYPE', 'COUNTY', 'PTRUCKS', 'ADT', 'AADT',

'DIRECTION', '85PCT', 'PRIORITY\_POINTS']

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writeheader()

for record in self.records:

writer.writerow({

'SECTION ID': record.section\_id,

'HIGHWAY': record.highway,

'SECTION': record.section,

'SECTION LENGTH': record.section\_length,

'SECTION DESCRIPTION': record.section\_description,

'Date': record.date,

'DESCRIPTION': record.description,

'GROUP': record.group,

'TYPE': record.type\_,

'COUNTY': record.county,

'PTRUCKS': record.ptrucks,

'ADT': record.adt,

'AADT': record.aadt,

'DIRECTION': record.direction,

'85PCT': record.pct85,

'PRIORITY\_POINTS': record.priority\_points

})

def add\_record(self, record\_data):

"""

Add a new TrafficRecord to the list of records.

:param record\_data: A dictionary containing the data for the new record.

"""

record = TrafficRecord(\*\*record\_data)

self.records.append(record)

def edit\_record(self, index, new\_data):

"""

Edit an existing TrafficRecord in the list of records.

:param index: The index of the record to edit.

:param new\_data: A dictionary containing the updated data for the record.

:return: True if the record was successfully edited, False otherwise.

"""

if 0 <= index < len(self.records):

for key, value in new\_data.items():

setattr(self.records[index], key, value)

return True

return False

def delete\_record(self, index):

"""

Delete a TrafficRecord from the list of records.

:param index: The index of the record to delete.

:return: True if the record was successfully deleted, False otherwise.

"""

if 0 <= index < len(self.records):

del self.records[index]

return True

return False

def get\_record(self, index):

"""

Retrieve a TrafficRecord from the list of records.

:param index: The index of the record to retrieve.

:return: The TrafficRecord object if found, None otherwise.

"""

if 0 <= index < len(self.records):

return self.records[index]

return None

def reload\_data(self):

"""

Reload the data from the CSV file.

"""

self.records = self.load\_data(self.filename)

def demonstrate\_polymorphism(self):

"""

Demonstrate polymorphic method calls using different types of records.

"""

sample\_records = [

TrafficRecord("1", "Highway1", "Section1", "10", "Description1", "2023-01-01", "Desc1", "Group1", "Type1", "County1", "10", "1000", "2000", "North", "85", "10"),

SpecialTrafficRecord("2", "Highway2", "Section2", "20", "Description2", "2023-01-02", "Desc2", "Group2", "Type2", "County2", "20", "2000", "3000", "South", "90", "20")

]

for record in sample\_records:

print(record.display())

**main.py**

# main.py

# Author: Meet Maheta

from traffic\_manager import TrafficManager

from presentation import display\_records, display\_menu, get\_user\_choice, get\_record\_details, display\_message

def main():

"""

The main function to run the traffic data management program.

It initializes the TrafficManager and provides a menu-driven interface for user interaction.

"""

host = "localhost" # e.g., "localhost"

database = "traffic" # e.g., "traffic\_db"

user = "mmb0702" # e.g., "root"

password = "System #1234" # e.g., "password123"

manager = TrafficManager(host, database, user, password) # Initialize the TrafficManager with the given database parameters

manager.create\_table() # Create the table if it does not exist

while True:

display\_menu() # Display the main menu

choice = get\_user\_choice() # Get the user's choice from the menu

if choice is None:

continue

if choice == 1:

display\_message("Reloading data...")

manager.reload\_data() # Reload data from the database

display\_message(f"Successfully loaded {len(manager.records)} records.")

elif choice == 2:

display\_message("Saving data...")

manager.save\_data() # Save data to the database

display\_message("Data saved successfully.")

elif choice == 3:

num\_records = int(input("Enter the number of records to display: "))

display\_records(manager.records, num\_records) # Display the specified number of records

elif choice == 4:

display\_message("Adding a new record...")

details = get\_record\_details() # Get new record details from the user

manager.add\_record({

'section\_id': details[0],

'highway': details[1],

'section': details[2],

'section\_length': float(details[3]),

'section\_description': details[4],

'date': details[5],

'description': details[6],

'group': details[7],

'type\_': details[8],

'county': details[9],

'ptrucks': details[10],

'adt': float(details[11]),

'aadt': float(details[12]),

'direction': details[13],

'pct85': details[14],

'priority\_points': details[15]

}) # Add the new record to the manager

display\_message("Record added successfully.")

elif choice == 5:

display\_message("Editing a record...")

id = int(input("Enter the index of the record to edit: "))

details = get\_record\_details() # Get updated record details from the user

if manager.edit\_record(id, {

'section\_id': details[0],

'highway': details[1],

'section': details[2],

'section\_length': float(details[3]),

'section\_description': details[4],

'date': details[5],

'description': details[6],

'group': details[7],

'type\_': details[8],

'county': details[9],

'ptrucks': details[10],

'adt': float(details[11]),

'aadt': float(details[12]),

'direction': details[13],

'pct85': details[14],

'priority\_points': details[15]

}):

display\_message("Record edited successfully.")

else:

display\_message("Record not found.")

elif choice == 6:

display\_message("Deleting a record...")

id = int(input("Enter the index of the record to delete: "))

if manager.delete\_record(id):

display\_message("Record deleted successfully.")

else:

display\_message("Record not found.")

elif choice == 7:

display\_message("Demonstrating polymorphism...")

manager.demonstrate\_polymorphism() # Demonstrate polymorphic method calls

elif choice == 0:

display\_message("Exiting program.")

break

else:

display\_message("Invalid choice. Please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**presentation.py**

# presentation.py

# Author: Meet Maheta

def display\_menu():

"""

Display the main menu to the user.

"""

print("\nProgram by Meet Maheta")

print("1. Reload data")

print("2. Save data")

print("3. Display records")

print("4. Add a new record")

print("5. Edit a record")

print("6. Delete a record")

print("7. Demonstrate polymorphism") # Added new option

print("0. Exit")

def get\_user\_choice():

"""

Get the user's menu choice.

Returns

-------

int or None

The user's choice as an integer, or None if the input is invalid.

"""

try:

choice = int(input("Enter your choice: "))

if choice in range(8): # Updated range to include the new option

return choice

except ValueError:

pass

print("Invalid choice, please try again.")

return None

def display\_records(records, num\_records):

"""

Display a specified number of traffic records.

Parameters

----------

records : list of TrafficRecord

The list of TrafficRecord objects to display.

num\_records : int

The number of records to display.

"""

for record in records[:num\_records]:

print(f"SECTION ID: {record.section\_id}, HIGHWAY: {record.highway}, SECTION: {record.section}, "

f"SECTION LENGTH: {record.section\_length}, SECTION DESCRIPTION: {record.section\_description}, "

f"DATE: {record.date}, DESCRIPTION: {record.description}, GROUP: {record.group}, TYPE: {record.type\_}, "

f"COUNTY: {record.county}, PTRUCKS: {record.ptrucks}, ADT: {record.adt}, AADT: {record.aadt}, "

f"DIRECTION: {record.direction}, 85PCT: {record.pct85}, PRIORITY\_POINTS: {record.priority\_points}")

def get\_record\_details():

"""

Get the details of a traffic record from the user.

Returns

-------

tuple

A tuple containing the details of the traffic record.

"""

section\_id = input("Enter SECTION ID: ")

highway = input("Enter HIGHWAY: ")

section = input("Enter SECTION: ")

section\_length = input("Enter SECTION LENGTH: ")

section\_description = input("Enter SECTION DESCRIPTION: ")

date = input("Enter Date: ")

description = input("Enter DESCRIPTION: ")

group = input("Enter GROUP: ")

type\_ = input("Enter TYPE: ")

county = input("Enter COUNTY: ")

ptrucks = input("Enter PTRUCKS: ")

adt = input("Enter ADT: ")

aadt = input("Enter AADT: ")

direction = input("Enter DIRECTION: ")

pct85 = input("Enter 85PCT: ")

priority\_points = input("Enter PRIORITY\_POINTS: ")

return section\_id, highway, section, section\_length, section\_description, date, description, group, type\_, county, ptrucks, adt, aadt, direction, pct85, priority\_points

def display\_message(message):

"""

Display a message to the user.

Parameters

----------

message : str

The message to display.

"""

print(message)

**presentation.py**

# presentation.py

# Author: Meet Maheta

def display\_menu():

"""

Display the main menu to the user.

"""

print("\nProgram by Meet Maheta")

print("1. Reload data")

print("2. Save data")

print("3. Display records")

print("4. Add a new record")

print("5. Edit a record")

print("6. Delete a record")

print("7. Demonstrate polymorphism") # Added new option

print("0. Exit")

def get\_user\_choice():

"""

Get the user's menu choice.

Returns

-------

int or None

The user's choice as an integer, or None if the input is invalid.

"""

try:

choice = int(input("Enter your choice: "))

if choice in range(8): # Updated range to include the new option

return choice

except ValueError:

pass

print("Invalid choice, please try again.")

return None

def display\_records(records, num\_records):

"""

Display a specified number of traffic records.

Parameters

----------

records : list of TrafficRecord

The list of TrafficRecord objects to display.

num\_records : int

The number of records to display.

"""

for record in records[:num\_records]:

print(f"SECTION ID: {record.section\_id}, HIGHWAY: {record.highway}, SECTION: {record.section}, "

f"SECTION LENGTH: {record.section\_length}, SECTION DESCRIPTION: {record.section\_description}, "

f"DATE: {record.date}, DESCRIPTION: {record.description}, GROUP: {record.group}, TYPE: {record.type\_}, "

f"COUNTY: {record.county}, PTRUCKS: {record.ptrucks}, ADT: {record.adt}, AADT: {record.aadt}, "

f"DIRECTION: {record.direction}, 85PCT: {record.pct85}, PRIORITY\_POINTS: {record.priority\_points}")

def get\_record\_details():

"""

Get the details of a traffic record from the user.

Returns

-------

tuple

A tuple containing the details of the traffic record.

"""

section\_id = input("Enter SECTION ID: ")

highway = input("Enter HIGHWAY: ")

section = input("Enter SECTION: ")

section\_length = input("Enter SECTION LENGTH: ")

section\_description = input("Enter SECTION DESCRIPTION: ")

date = input("Enter Date: ")

description = input("Enter DESCRIPTION: ")

group = input("Enter GROUP: ")

type\_ = input("Enter TYPE: ")

county = input("Enter COUNTY: ")

ptrucks = input("Enter PTRUCKS: ")

adt = input("Enter ADT: ")

aadt = input("Enter AADT: ")

direction = input("Enter DIRECTION: ")

pct85 = input("Enter 85PCT: ")

priority\_points = input("Enter PRIORITY\_POINTS: ")

return section\_id, highway, section, section\_length, section\_description, date, description, group, type\_, county, ptrucks, adt, aadt, direction, pct85, priority\_points

def display\_message(message):

"""

Display a message to the user.

Parameters

----------

message : str

The message to display.

"""

print(message)

**record\_base.py**

# record\_base.py

# Author: Meet Maheta

class RecordBase:

"""

A base class representing a traffic record.

Attributes:

-----------

section\_id : str

The ID of the section.

highway : str

The name of the highway.

section : str

The section of the highway.

section\_length : float

The length of the section.

section\_description : str

The description of the section.

date : str

The date of the record.

description : str

The description of the record.

group : str

The group associated with the record.

type\_ : str

The type of the record.

county : str

The county where the section is located.

ptrucks : str

The percentage of trucks in the traffic.

adt : float

The average daily traffic.

aadt : float

The annual average daily traffic.

direction : str

The direction of the traffic.

pct85 : str

The 85th percentile speed.

priority\_points : str

The priority points of the section.

"""

def \_\_init\_\_(self, section\_id, highway, section, section\_length, section\_description, date, description, group, type\_, county, ptrucks, adt, aadt, direction, pct85, priority\_points):

"""

Initialize a RecordBase object with the provided attributes.

Parameters:

-----------

section\_id : str

The ID of the section.

highway : str

The name of the highway.

section : str

The section of the highway.

section\_length : float

The length of the section.

section\_description : str

The description of the section.

date : str

The date of the record.

description : str

The description of the record.

group : str

The group associated with the record.

type\_ : str

The type of the record.

county : str

The county where the section is located.

ptrucks : str

The percentage of trucks in the traffic.

adt : float

The average daily traffic.

aadt : float

The annual average daily traffic.

direction : str

The direction of the traffic.

pct85 : str

The 85th percentile speed.

priority\_points : str

The priority points of the section.

"""

self.section\_id = section\_id

self.highway = highway

self.section = section

self.section\_length = section\_length

self.section\_description = section\_description

self.date = date

self.description = description

self.group = group

self.type\_ = type\_

self.county = county

self.ptrucks = ptrucks

self.adt = adt

self.aadt = aadt

self.direction = direction

self.pct85 = pct85

self.priority\_points = priority\_points

def display(self):

"""

Return a string representation of the traffic record.

Returns:

--------

str

A string containing all the attributes of the traffic record.

"""

return f"SECTION ID: {self.section\_id}, HIGHWAY: {self.highway}, SECTION: {self.section}, SECTION LENGTH: {self.section\_length}, SECTION DESCRIPTION: {self.section\_description}, DATE: {self.date}, DESCRIPTION: {self.description}, GROUP: {self.group}, TYPE: {self.type\_}, COUNTY: {self.county}, PTRUCKS: {self.ptrucks}, ADT: {self.adt}, AADT: {self.aadt}, DIRECTION: {self.direction}, 85PCT: {self.pct85}, PRIORITY\_POINTS: {self.priority\_points}"

**record.py**

# record.py

# Author: Meet Maheta

from record\_base import RecordBase

class TrafficRecord(RecordBase):

"""

A subclass of RecordBase representing a specific type of traffic record.

"""

def \_\_init\_\_(self, section\_id, highway, section, section\_length, section\_description, date, description, group, type\_, county, ptrucks, adt, aadt, direction, pct85, priority\_points):

"""

Initialize a TrafficRecord object with the provided attributes by calling the superclass initializer.

Parameters:

-----------

section\_id : str

The ID of the section.

highway : str

The name of the highway.

section : str

The section of the highway.

section\_length : float

The length of the section.

section\_description : str

The description of the section.

date : str

The date of the record.

description : str

The description of the record.

group : str

The group associated with the record.

type\_ : str

The type of the record.

county : str

The county where the section is located.

ptrucks : str

The percentage of trucks in the traffic.

adt : float

The average daily traffic.

aadt : float

The annual average daily traffic.

direction : str

The direction of the traffic.

pct85 : str

The 85th percentile speed.

priority\_points : str

The priority points of the section.

"""

super().\_\_init\_\_(section\_id, highway, section, section\_length, section\_description, date, description, group, type\_, county, ptrucks, adt, aadt, direction, pct85, priority\_points)

def display(self):

"""

Return a string representation of the traffic record.

Returns:

--------

str

A string containing the type of the record and all its attributes.

"""

return f"Traffic Record - {super().display()}"

**special\_traffic\_record.py**

# special\_traffic\_record.py

# Author: Meet Maheta

from record\_base import RecordBase

class SpecialTrafficRecord(RecordBase):

"""

A subclass of RecordBase representing a special type of traffic record.

"""

def \_\_init\_\_(self, section\_id, highway, section, section\_length, section\_description, date, description, group, type\_, county, ptrucks, adt, aadt, direction, pct85, priority\_points):

"""

Initialize a SpecialTrafficRecord object with the provided attributes by calling the superclass initializer.

Parameters:

-----------

section\_id : str

The ID of the section.

highway : str

The name of the highway.

section : str

The section of the highway.

section\_length : float

The length of the section.

section\_description : str

The description of the section.

date : str

The date of the record.

description : str

The description of the record.

group : str

The group associated with the record.

type\_ : str

The type of the record.

county : str

The county where the section is located.

ptrucks : str

The percentage of trucks in the traffic.

adt : float

The average daily traffic.

aadt : float

The annual average daily traffic.

direction : str

The direction of the traffic.

pct85 : str

The 85th percentile speed.

priority\_points : str

The priority points of the section.

"""

super().\_\_init\_\_(section\_id, highway, section, section\_length, section\_description, date, description, group, type\_, county, ptrucks, adt, aadt, direction, pct85, priority\_points)

def display(self):

"""

Return a string representation of the special traffic record.

Returns:

--------

str

A string containing the type of the record and all its attributes in a special format.

"""

return f"Special Traffic Record - {super().display()} [SPECIAL FORMAT]"

**test\_business.py**

# test\_business.py

# Author: Meet Maheta

import unittest

from traffic\_manager import TrafficManager

class TestTrafficManager(unittest.TestCase):

"""

Unit tests for the TrafficManager class.

"""

def setUp(self):

"""

Set up the test environment by initializing a TrafficManager instance

and clearing the traffic\_volumes table.

"""

self.manager = TrafficManager('localhost', 'traffic', 'mmb0702', 'System #1234')

self.manager.create\_table()

if self.manager.conn:

cursor = self.manager.conn.cursor()

cursor.execute("DELETE FROM traffic\_volumes")

self.manager.conn.commit()

def test\_add\_record(self):

"""

Test the add\_record method to ensure it correctly adds a new record.

"""

initial\_count = len(self.manager.load\_data())

self.manager.add\_record({

'section\_id': '9999', 'highway': 'H', 'section': 'S', 'section\_length': 1.0,

'section\_description': 'Desc', 'date': '2023-01-01', 'description': 'Desc',

'group': 'G', 'type\_': 'T', 'county': 'C', 'ptrucks': 'P', 'adt': 1000.0,

'aadt': 2000.0, 'direction': 'D', 'pct85': '85', 'priority\_points': 'PP'

})

self.assertEqual(len(self.manager.load\_data()), initial\_count + 1)

def test\_reload\_data(self):

"""

Test the reload\_data method to ensure it correctly reloads the data from the database.

"""

self.manager.add\_record({

'section\_id': '9999', 'highway': 'H', 'section': 'S', 'section\_length': 1.0,

'section\_description': 'Desc', 'date': '2023-01-01', 'description': 'Desc',

'group': 'G', 'type\_': 'T', 'county': 'C', 'ptrucks': 'P', 'adt': 1000.0,

'aadt': 2000.0, 'direction': 'D', 'pct85': '85', 'priority\_points': 'PP'

})

self.manager.reload\_data()

self.assertGreater(len(self.manager.records), 0)

def test\_edit\_record(self):

"""

Test the edit\_record method to ensure it correctly edits an existing record.

"""

self.manager.add\_record({

'section\_id': '9999', 'highway': 'H', 'section': 'S', 'section\_length': 1.0,

'section\_description': 'Desc', 'date': '2023-01-01', 'description': 'Desc',

'group': 'G', 'type\_': 'T', 'county': 'C', 'ptrucks': 'P', 'adt': 1000.0,

'aadt': 2000.0, 'direction': 'D', 'pct85': '85', 'priority\_points': 'PP'

})

cursor = self.manager.conn.cursor()

cursor.execute("SELECT LAST\_INSERT\_ID()")

id = cursor.fetchone()[0]

self.manager.edit\_record(id, {

'section\_id': '9999', 'highway': 'H', 'section': 'S', 'section\_length': 1.0,

'section\_description': 'Desc', 'date': '2023-01-01', 'description': 'Desc',

'group': 'G', 'type\_': 'T', 'county': 'C', 'ptrucks': 'P', 'adt': 2000.0,

'aadt': 2000.0, 'direction': 'D', 'pct85': '85', 'priority\_points': 'PP'

})

self.assertEqual(self.manager.get\_record(id).adt, 2000.0)

def test\_delete\_record(self):

"""

Test the delete\_record method to ensure it correctly deletes a record.

"""

self.manager.add\_record({

'section\_id': '9999', 'highway': 'H', 'section': 'S', 'section\_length': 1.0,

'section\_description': 'Desc', 'date': '2023-01-01', 'description': 'Desc',

'group': 'G', 'type\_': 'T', 'county': 'C', 'ptrucks': 'P', 'adt': 1000.0,

'aadt': 2000.0, 'direction': 'D', 'pct85': '85', 'priority\_points': 'PP'

})

cursor = self.manager.conn.cursor()

cursor.execute("SELECT LAST\_INSERT\_ID()")

id = cursor.fetchone()[0]

self.manager.delete\_record(id)

self.assertIsNone(self.manager.get\_record(id))

def test\_demonstrate\_polymorphism(self):

"""

Test the demonstrate\_polymorphism method to ensure it displays the correct output.

"""

self.manager.demonstrate\_polymorphism()

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()