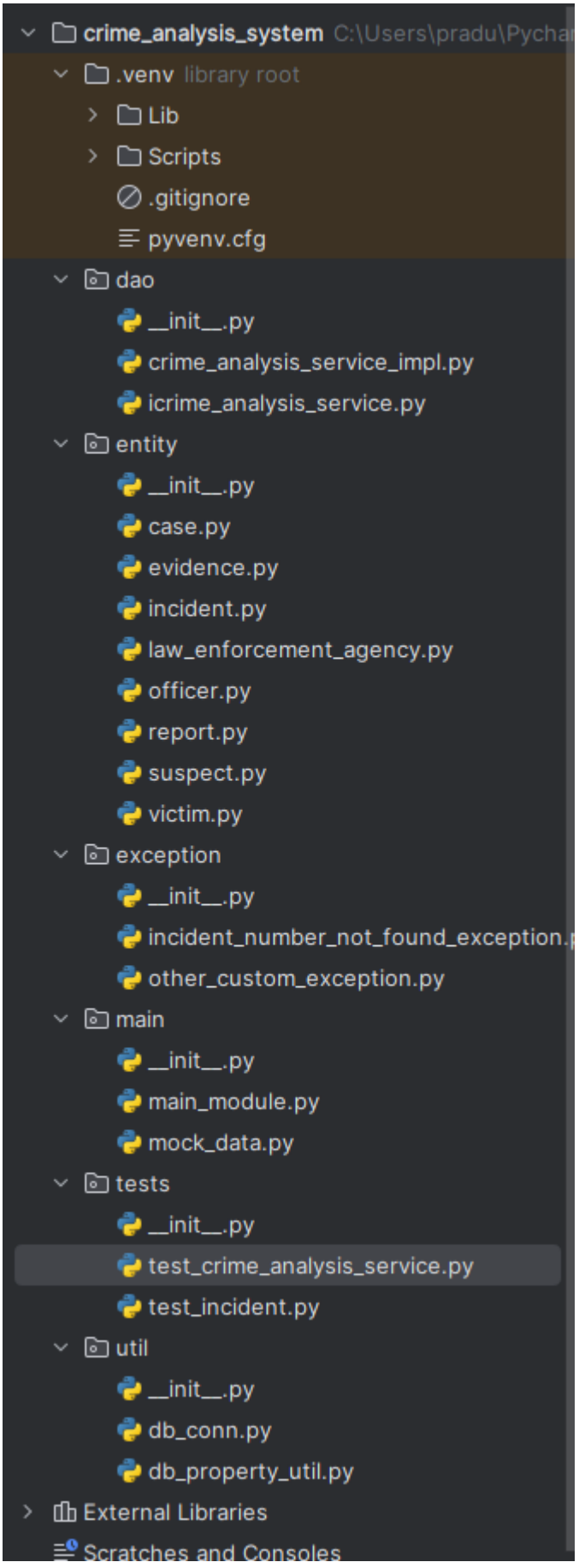
Case Study C.A.R.S

Case Study - 3 | Crime Reporting System | Pradum singh

- ***Directory Structure***

```
crime-analysis-reporting-system
|-- src
|   |-- entity
|   |   |-- Incidents.py
|   |   |-- Victims.py
|   |   |-- Suspects.py
|   |   |-- LawEnforcementAgencies.py
|   |   |-- Officers.py
|   |   |-- Evidence.py
|   |   |-- Reports.py
|   |   |-- Case.py
|   |
|   |-- dao
|   |   |-- ICrimeAnalysisService.py
|   |   |-- CrimeAnalysisServiceImpl.py
|   |
|   |-- exception
|   |   |-- CustomExceptions.py
|   |
|   |-- util
|   |   |-- DBConnection.py
|   |   |-- PropertyUtil.py
|   |
|   |-- main
|   |   |-- MainModule.py
|   |
|   |-- tests
|   |   |-- test_CrimeAnalysisService.py
|   |
|   |-- resources
|   |   |-- db_properties.ini
```


crime_analysis_system C:\Users\pradu\Pycha
.venv library root
Lib
Scripts
.gitignore
pyvenv.cfg
dao
__init__.py
crime_analysis_service_impl.py
icrime_analysis_service.py
entity
__init__.py
case.py
evidence.py
incident.py
law_enforcement_agency.py
officer.py
report.py
suspect.py
victim.py
exception
__init__.py
incident_number_not_found_exception.
other_custom_exception.py
main
__init__.py
main_module.py
mock_data.py
tests
__init__.py
test_crime_analysis_service.py
test_incident.py
util
__init__.py
db_conn.py
db_property_util.py
External Libraries
Scratches and Consoles

- ***Task 1: Entity Classes (in the "entity" package)***
  - *Create the following entity classes:*

1. *Incident*

```python
class Incident:
    def __init__(self, incident_id, incident_type, incident_date, location, description, status, victim_id, suspect_id):
        self.incident_id = incident_id
        self.incident_type = incident_type
        self.incident_date = incident_date
        self.location = location
        self.description = description
        self.status = status
        self.victim_id = victim_id
        self.suspect_id = suspect_id
        self.evidences = []  # Initialize evidences as an empty list
```

2. *Victim*

```python
class Victim:
    def __init__(self, victim_id, first_name, last_name, date_of_birth, gender, contact_information):
        self.victim_id = victim_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.gender = gender
        self.contact_information = contact_information
        self.incidents = []  # Initialize incidents as an empty list
```

3. *Suspect*

```python
class Suspect:
    def __init__(self, suspect_id, first_name, last_name, date_of_birth, gender, contact_information):
        self.suspect_id = suspect_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.gender = gender
        self.contact_information = contact_information
        self.incidents = []  # Initialize incidents as an empty list
```

4. *Law Enforcement Agency*

```python
class LawEnforcementAgency:
    def __init__(self, agency_id, agency_name, jurisdiction, contact_information, officers=None):
        self.agency_id = agency_id
        self.agency_name = agency_name
        self.jurisdiction = jurisdiction
        self.contact_information = contact_information
        self.officers = officers if officers is not None else []  # Initialize officers as an empty list if not provided
```

5. *Officer*

```python
class Officer:
    def __init__(self, officer_id, first_name, last_name, badge_number, rank, contact_info, agency_id):
        self.officer_id = officer_id
        self.first_name = first_name
        self.last_name = last_name
        self.badge_number = badge_number
        self.rank = rank
        self.contact_info = contact_info
        self.agency_id = agency_id
        self.reports = []  # Initialize reports as an empty list
```

6. *Evidence*

```python
class Evidence:
    def __init__(self, evidence_id, description, location_found, incident_id):
        self.evidence_id = evidence_id
        self.description = description
        self.location_found = location_found
        self.incident_id = incident_id

    def __str__(self):
        return f"EvidenceID: {self.evidence_id}, Description: {self.description}, LocationFound: {self.location_found}, IncidentID: {self.incident_id}"
```

7. *Report*

```python
class Report:
    def __init__(self, report_id, incident_id, reporting_officer, report_date, report_details, status):
        self.report_id = report_id
        self.incident_id = incident_id
        self.reporting_officer = reporting_officer
        self.report_date = report_date
        self.report_details = report_details
        self.status = status
```

- *Declare private variables in each class corresponding to the schema.*
- *Implement default and parameterized constructors in each class.*
- *Implement getters and setters for the private variables.*

- ***Task 2 : Service Provider Interface (in the “dao” package)***

1. `createIncident(Incident incident): boolean`
2. `updateIncidentStatus(Status status, int incidentId): boolean`
3. `getIncidentsInDateRange(Date startDate, Date endDate): Collection<Incident>`
4. `searchIncidents(IncidentType criteria): Collection<Incident>`
5. `generateIncidentReport(Incident incident): Report`
6. `createCase(String caseDescription, Collection<Incident> incidents): Case`
7. `getCaseDetails(int caseId): Case`
8. `updateCaseDetails(Case case): boolean`
9. `getAllCases(): List<Case>`

```python
from abc import ABC, abstractmethod
from typing import Collection
from entity.report import Report
from entity.case import Case

2 usages
class ICrimeAnalysisService(ABC):

    @abstractmethod
    def create_incident(self, incident) -> bool:
        pass

    @abstractmethod
    def update_incident_status(self, status, incident_id) -> bool:
        pass

    @abstractmethod
    def get_incidents_in_date_range(self, start_date, end_date) -> Collection:
        pass

    1 usage
    @abstractmethod
    def search_incidents(self, criteria) -> Collection:
        pass

    1 usage
    @abstractmethod
    def generate_incident_report(self, incident) -> Report:
        pass

    @abstractmethod
    def create_case(self, case_description, incidents) -> Case:
        pass
```

- ***Task 3: Connect to SQL Database (in the "util" package)***
  - *Create a utility class `DBConnection` with a static variable `connection` of type `Connection`.*
  - *Implement a static method `getConnection()` in `DBConnection` that returns the database connection.*

```python
''' This class helps your program connect to a database. In simpler terms, imagine it as a tool that lets your program talk to the database and ask for information.
In this tool, there's a special room (variable) where
the program can go and connect to the database.'''

import mysql.connector
from mysql.connector import Error
from util.db_property_util import PropertyUtil

4 usages
class DBConnection:
    connection = None

    @staticmethod
    def get_connection():
        if DBConnection.connection is None:
            try:
                # Use the information from PropertyUtil to connect to the database
                connection_string = PropertyUtil.get_property_string()
                DBConnection.connection = mysql.connector.connect(**connection_string)

                if DBConnection.connection.is_connected():
                    print("Connected to MySQL database")
                else:
                    print("Failed to connect to MySQL database")

            except Error as e:
                print(f"Error: {e}")

        return DBConnection.connection
```

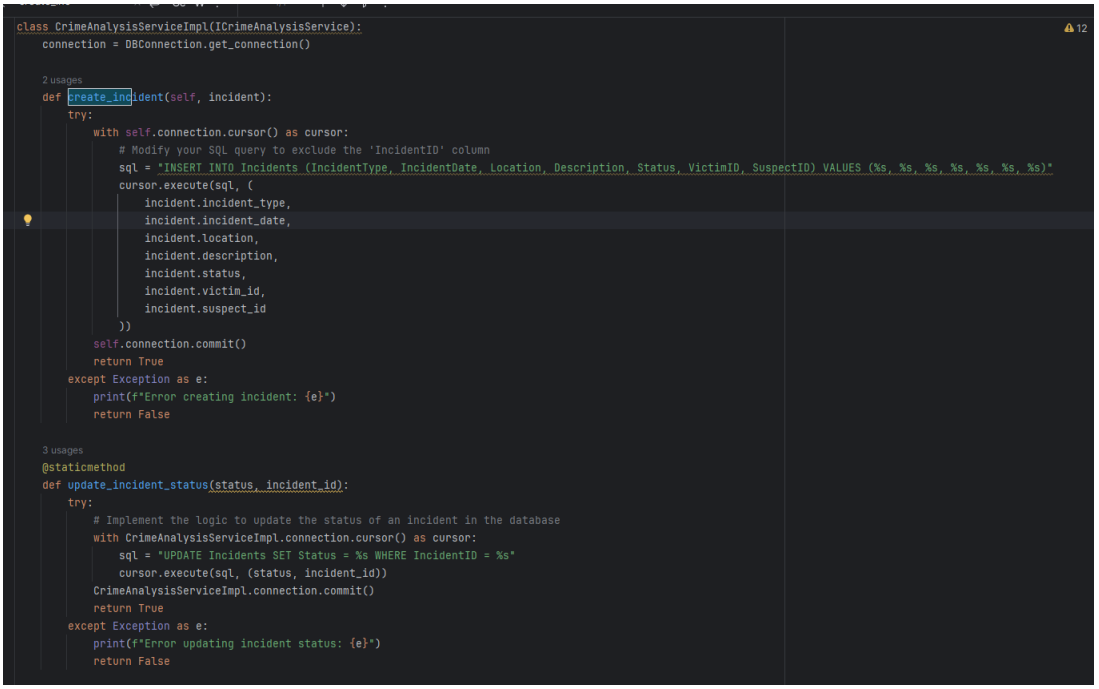
- *Create a utility class `PropertyUtil` with a static method `getPropertyString()` to read connection details from a property file and return a connection string.*

```python
'''This class helps your program get the details it needs
to connect to the database. In simpler terms, think of
it as a helper that reads a note (property file)
containing important details like the name of the
 house (database), the key to the house (username), etc.
 It then  hands over this information to the DBConnection tool.'''
# main_module.py
import mysql.connector
from mysql.connector import Error
import configparser

5 usages
class DBConnection:
    connection = None

    1 usage
    @staticmethod
    def get_connection():
        if DBConnection.connection is None:
            try:
                # Use the information from PropertyUtil to connect to the database
                connection_string = PropertyUtil.get_property_string()
                DBConnection.connection = mysql.connector.connect(**connection_string)

                if DBConnection.connection.is_connected():
                    print("Connected to MySQL database")
                else:
                    print("Failed to connect to MySQL database")

            except Error as e:
                print(f"Error: {e}")

        return DBConnection.connection
```

- ++

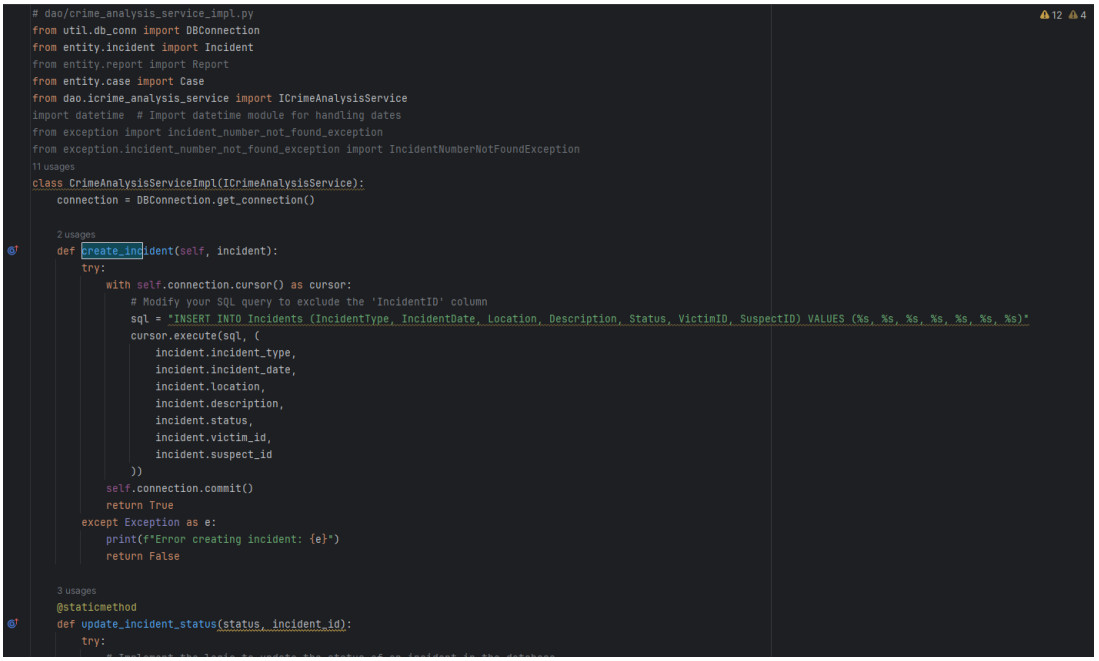
```
3 usages
class PropertyUtil:
    2 usages
    @staticmethod
    def get_property_string():
        # Set the provided details directly in the method
        connection_string = {
            'host': 'localhost',
            'user': 'root',
            'password': 'Batman@123#',
            'database': 'crime_analysis_system',
            'port': '3306'
        }

        return connection_string

1 usage
def main():
    # Establish a connection to the database
    connection = DBConnection.get_connection()

    # Your application logic goes here
    # ...

if __name__ == "__main__":
    main()
```

- ***Task 4: Service Implementation (in the "dao" package)***
  - *Create a class named* `CrimeAnalysisServiceImpl` *in the "dao" package.*
  - *Add a static variable named* `connection` *of type* `Connection`.
  - *In the constructor, initialize* `connection` *by invoking* `getConnection()` *from* `DBConnection`.



```
class CrimeAnalysisServiceImpl(ICrimeAnalysisService):
    connection = DBConnection.get_connection()

    2 usages
    def create_incident(self, incident):
        try:
            with self.connection.cursor() as cursor:
                # Modify your SQL query to exclude the 'IncidentID' column
                sql = "INSERT INTO Incidents (IncidentType, IncidentDate, Location, Description, Status, VictimID, SuspectID) VALUES (%s, %s, %s, %s, %s, %s, %s)"
                cursor.execute(sql, (
                    incident.incident_type,
                    incident.incident_date,
                    incident.location,
                    incident.description,
                    incident.status,
                    incident.victim_id,
                    incident.suspect_id
                ))
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error creating incident: {e}")
            return False

    3 usages
    @staticmethod
    def update_incident_status(status, incident_id):
        try:
            # Implement the logic to update the status of an incident in the database
            with CrimeAnalysisServiceImpl.connection.cursor() as cursor:
                sql = "UPDATE Incidents SET Status = %s WHERE IncidentID = %s"
                cursor.execute(sql, (status, incident_id))
            CrimeAnalysisServiceImpl.connection.commit()
            return True
        except Exception as e:
            print(f"Error updating incident status: {e}")
            return False
```

- *Provide implementations for all the methods defined in the* `ICrimeAnalysisService` *interface.*



```
# dao/crime_analysis_service_impl.py
from util.db_conn import DBConnection
from entity.incident import Incident
from entity.report import Report
from entity.case import Case
from dao.icrime_analysis_service import ICrimeAnalysisService
import datetime  # Import datetime module for handling dates
from exception import incident_number_not_found_exception
from exception.incident_number_not_found_exception import IncidentNumberNotFoundException
11 usages
class CrimeAnalysisServiceImpl(ICrimeAnalysisService):
    connection = DBConnection.get_connection()

    2 usages
    def create_incident(self, incident):
        try:
            with self.connection.cursor() as cursor:
                # Modify your SQL query to exclude the 'IncidentID' column
                sql = "INSERT INTO Incidents (IncidentType, IncidentDate, Location, Description, Status, VictimID, SuspectID) VALUES (%s, %s, %s, %s, %s, %s, %s)"
                cursor.execute(sql, (
                    incident.incident_type,
                    incident.incident_date,
                    incident.location,
                    incident.description,
                    incident.status,
                    incident.victim_id,
                    incident.suspect_id
                ))
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error creating incident: {e}")
            return False

    3 usages
    @staticmethod
    def update_incident_status(status, incident_id):
        try:
```

```python
@staticmethod
def update_incident_status(status, incident_id):
    try:
        # Implement the logic to update the status of an incident in the database
        with CrimeAnalysisServiceImpl.connection.cursor() as cursor:
            sql = "UPDATE Incidents SET Status = %s WHERE IncidentID = %s"
            cursor.execute(sql, (status, incident_id))
        CrimeAnalysisServiceImpl.connection.commit()
        return True
    except Exception as e:
        print(f"Error updating incident status: {e}")
        return False

1 usage
@staticmethod
def get_incidents_in_date_range(start_date, end_date):
    try:
        # Implement the logic to get a list of incidents within a date range from the database
        with CrimeAnalysisServiceImpl.connection.cursor() as cursor:
            sql = "SELECT * FROM Incidents WHERE IncidentDate BETWEEN %s AND %s"
            cursor.execute(sql, (start_date, end_date))
            incidents = cursor.fetchall()
            return [Incident(*incident) for incident in incidents]
    except Exception as e:
        print(f"Error getting incidents in date range: {e}")
        return []

1 usage
@staticmethod
def get_incident_details(incident_number):
    try:
        # Implement the logic to get incident details from the database
        with CrimeAnalysisServiceImpl.connection.cursor() as cursor:
            sql = "SELECT * FROM Incidents WHERE IncidentID = %s"
            cursor.execute(sql, (incident_number,))
            incident_details = cursor.fetchone()

            if incident_details:
                return Incident(*incident_details)
```

```
    @staticmethod
    def get_incident_details(incident_number):
        try:
            # Implement the logic to get incident details from the database
            with CrimeAnalysisServiceImpl.connection.cursor() as cursor:
                sql = "SELECT * FROM Incidents WHERE IncidentID = %s"
                cursor.execute(sql, (incident_number,))
                incident_details = cursor.fetchone()

                if incident_details:
                    return Incident(*incident_details)
                else:
                    # If incident not found, raise a generic exception
                    raise Exception(f"Incident with number {incident_number} not found.")
        except Exception as e:
            # Handle the exception
            print(f"Exception: {e}")
            return None
    @staticmethod
    def update_case_details(case):
        try:
            # Implement the logic to update case details in the database
            with CrimeAnalysisServiceImpl.connection.cursor() as cursor:
                sql = "UPDATE Cases SET CaseDescription = %s WHERE CaseID = %s"
                cursor.execute(sql, (case.case_description, case.case_id))
            CrimeAnalysisServiceImpl.connection.commit()
            return True
        except Exception as e:
            print(f"Error updating case details: {e}")
            return False

    @staticmethod
    def get_all_cases():
        try:
            # Implement the logic to get a list of all cases from the database
            with CrimeAnalysisServiceImpl.connection.cursor() as cursor:
                sql = "SELECT * FROM Cases"
                cursor.execute(sql)
                cases = cursor.fetchall()
```

```python
@staticmethod
def get_all_cases():
    try:
        # Implement the logic to get a list of all cases from the database
        with CrimeAnalysisServiceImpl.connection.cursor() as cursor:
            sql = "SELECT * FROM Cases"
            cursor.execute(sql)
            cases = cursor.fetchall()
            return [Case(*case) for case in cases]
    except Exception as e:
        print(f"Error getting all cases: {e}")
        return []

@staticmethod
def search_incidents(criteria):
    try:
        # Implement the logic to search incidents based on criteria in the database
        # Replace the following line with your implementation
        return []
    except Exception as e:
        print(f"Error searching incidents: {e}")
        return []

@staticmethod
def generate_incident_report(incident):
    try:
        # Implement the logic to generate an incident report in the database
        # Replace the following line with your implementation
        return None
    except Exception as e:
        print(f"Error generating incident report: {e}")
        return None

@staticmethod
def create_case(case_description, incidents):
    try:
        # Implement the logic to create a new case and associate it with incidents in the database
        # Replace the following line with your implementation
```

- ***Task 5: Exception Handling (in the “exception package)***

```python
1 usage
class IncidentNumberNotFoundException(Exception):
    def __init__(self, incident_number):
        self.incident_number = incident_number
        super().__init__(f"Incident with number {incident_number} not found.")
```

  - *Create a package named* `exception`.
  - *Define the following custom exceptions:*
    1. `IncidentNumberNotFoundException`
  - *Throw these exceptions in methods whenever needed.*
  - *Handle these exceptions in the main method.*

```
        return None

    @staticmethod
    def create_case(case_description, incidents):
        try:
            # Implement the logic to create a new case and associate it with incidents in the database
            # Replace the following line with your implementation
            return None
        except Exception as e:
            print(f"Error creating case: {e}")
            return None

    @staticmethod
    def get_case_details(case_id):
        try:
            # Implement the logic to get details of a specific case from the database
            # Replace the following line with your implementation
            return None
        except Exception as e:
            print(f"Error getting case details: {e}")
            return None
```

- ***Task 6: Main method (in the “main” package)***
  - *Create a class named* `MainModule` *with the main method.*

```
# main/main_module.py
from dao.crime_analysis_service_impl import CrimeAnalysisServiceImpl
from entity.incident import Incident
from exception import incident_number_not_found_exception

1 usage
class MainModule:
    1 usage
    @staticmethod
    def main():
        # Create an instance of the service implementation
        crime_analysis_service = CrimeAnalysisServiceImpl()

        # Create an incident
        # Create an incident without specifying incident_id
        # Create an incident without specifying incident_id
        new_incident = Incident(
            incident_type="Robbery",
            incident_date="2024-02-05",
            location="Latitude: 40.7128, Longitude: -74.0060",
            description="A robbery occurred at a local store.",
            status="Open",
            victim_id=101,
            suspect_id=201
        )

        # Create incident in the database
        success = crime_analysis_service.create_incident(new_incident)
        print(f"Incident created successfully: {success}")

        # Update incident status
        incident_id_to_update = 1
        new_status = "Closed"
        success = crime_analysis_service.update_incident_status(new_status, incident_id_to_update)
        print(f"Incident status updated successfully: {success}")

        # Get incidents within a date range
        start_date = "2024-01-01"
        end_date = "2024-12-31"
        incidents_in_date_range = crime_analysis_service.get_incidents_in_date_range(start_date, end_date)
        print("Incidents within date range:")
```

  - *Demonstrate the functionalities in a menu-driven application.*

- *Trigger all the methods in the service implementation class (`CrimeAnalysisServiceImpl`).*

- ***Task 7: Unit Testing***
  - *Incident Creation*
  - *Incident Status Update*
  - test_crime_analysis_service
    -

```python
# tests/test_crime_analysis_service.py
import unittest
from dao.crime_analysis_service_impl import CrimeAnalysisServiceImpl
from entity.incident import Incident
from exception import incident_number_not_found_exception

class TestCrimeAnalysisService(unittest.TestCase):

    def setUp(self):
        # Create a CrimeAnalysisServiceImpl instance
        self.crime_analysis_service = CrimeAnalysisServiceImpl()

    def test_create_incident(self):
        # Test incident creation
        new_incident = Incident(
            incident_type="Robbery",
            incident_date="2024-02-05",
            location="Location",
            description="Description",
            status="Open",
            victim_id=101,
            suspect_id=201
        )

        success = self.crime_analysis_service.create_incident(new_incident)

        # Check if the incident is created successfully
        self.assertTrue(success)

        # Check if the incident attributes are accurate
        self.assertEqual( first: "Robbery", new_incident.incident_type)
        self.assertEqual( first: "2024-02-05", new_incident.incident_date)
        self.assertEqual( first: "Location", new_incident.location)
        self.assertEqual( first: "Description", new_incident.description)
        self.assertEqual( first: "Open", new_incident.status)
        self.assertEqual( first: 101, new_incident.victim_id)
        self.assertEqual( first: 201, new_incident.suspect_id)

    def test_update_incident_status(self):
```

  - ++
    -

```
def test_update_incident_status(self):
    # Create a sample incident
    incident = Incident(
        incident_id=1,
        incident_type="Robbery",
        incident_date="2024-02-05",
        location="Location",
        description="Description",
        status="Open",
        victim_id=101,
        suspect_id=201
    )

    # Test incident status update
    success = self.crime_analysis_service.update_incident_status( status: "Closed", incident)

    # Check if the status is updated successfully
    self.assertTrue(success)

    # Check if the incident's status is updated correctly
    self.assertEqual( first: "Closed", incident.status)

def test_invalid_status_update(self):
    # Create a sample incident
    incident = Incident(
        incident_id=1,
        incident_type="Robbery",
        incident_date="2024-02-05",
        location="Location",
        description="Description",
        status="Open",
        victim_id=101,
        suspect_id=201
    )

    # Test invalid incident status update
    with self.assertRaises(IncidentNumberNotFoundException):
        self.crime_analysis_service.update_incident_status( status: "InvalidStatus", incident)
```

- ++

```
    )

    # Test incident status update
    success = self.crime_analysis_service.update_incident_status( status: "Closed", incident)

    # Check if the status is updated successfully
    self.assertTrue(success)

    # Check if the incident's status is updated correctly
    self.assertEqual( first: "Closed", incident.status)

def test_invalid_status_update(self):
    # Create a sample incident
    incident = Incident(
        incident_id=1,
        incident_type="Robbery",
        incident_date="2024-02-05",
        location="Location",
        description="Description",
        status="Open",
        victim_id=101,
        suspect_id=201
    )

    # Test invalid incident status update
    with self.assertRaises(IncidentNumberNotFoundException):
        self.crime_analysis_service.update_incident_status( status: "InvalidStatus", incident)

if __name__ == '__main__':
    unittest.main()
```