Case Study: Crime Analysis and Reporting System (C.A.R.S.)

Key Functionalities:
The primary objective of this project is to develop a comprehensive **Crime Analysis and Reporting System (CARS)** that addresses the above-mentioned challenges and provides law enforcement agencies with a robust, user-friendly, and secure platform for crime data management and reporting.
1. Schema design:

Entities:
1. Incidents: • IncidentID (Primary Key)

• IncidentType (e.g., Robbery, Homicide, Theft)

• IncidentDate

• Location (Geospatial Data: Latitude and Longitude)

• Description

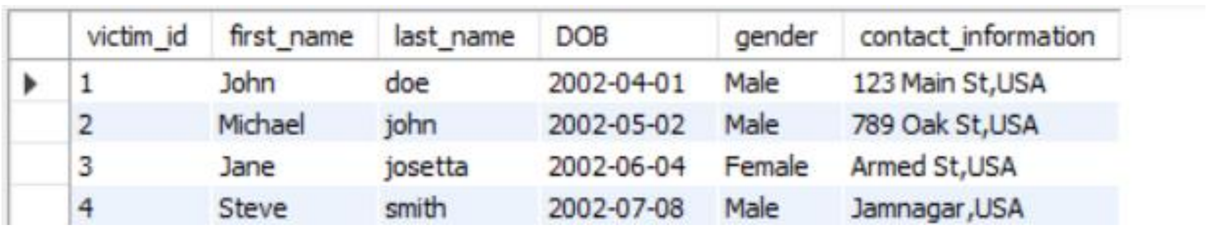
• Status (e.g., Open, Closed, Under Investigation)

• VictimID (Foreign Key, linking to Victims)

• SuspectId(Foreign Key, Linking to Suspect)

```
create database Crime_Reporting;

use Crime_Reporting;


Create table Victims(

victim_id int primary key,

first_name varchar(10),

last_name varchar(10),

DOB date,

gender varchar(15),

contact_information text

);


insert into Victims (victim_id,first_name,last_name,DOB,gender,contact_information)

VALUES
```

```
(1,'John','doe','2002-04-01','Male','123 Main St,USA'),
(2,'Michael','john','2002-05-02','Male','789 Oak St,USA'),
(3,'Jane','josetta','2002-06-04','Female','Armed St,USA'),
(4,'Steve','smith','2002-07-08','Male','Jamnagar,USA');
```



victim_id	first_name	last_name	DOB	gender	contact_information
1	John	doe	2002-04-01	Male	123 Main St,USA
2	Michael	john	2002-05-02	Male	789 Oak St,USA
3	Jane	josetta	2002-06-04	Female	Armed St,USA
4	Steve	smith	2002-07-08	Male	Jamnagar,USA

2. Victims:

• VictimID (Primary Key)

• FirstName

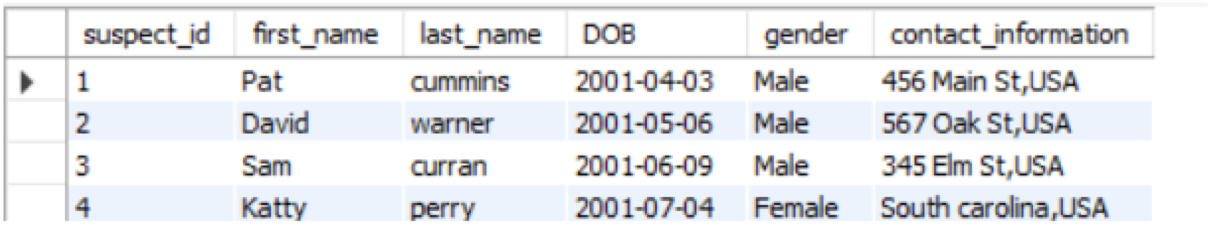
• LastName

• DateOfBirth

• Gender

• Contact Information (e.g., Address, Phone Number)

```
Create table Suspects(
suspect_id int primary key,
first_name varchar(10),
last_name varchar(10),
DOB date,
gender varchar(15),
contact_information text
);

insert into Suspects(suspect_id,first_name,last_name,DOB,gender,contact_information)
VALUES
(1,'Pat','cummins','2001-04-03','Male','456 Main St,USA'),
(2,'David','warner','2001-05-06','Male','567 Oak St,USA'),
(3,'Sam','curran','2001-06-09','Male','345 Elm St,USA'),
(4,'Katty','perry','2001-07-04','Female','South carolina,USA');
```



suspect_id	first_name	last_name	DOB	gender	contact_information
1	Pat	cummins	2001-04-03	Male	456 Main St,USA
2	David	warner	2001-05-06	Male	567 Oak St,USA
3	Sam	curran	2001-06-09	Male	345 Elm St,USA
4	Katty	perry	2001-07-04	Female	South carolina,USA

3. Suspects:

- SuspectID (Primary Key)
- FirstName
- LastName
- DateOfBirth
- Gender
- Contact Information

```
Create table Incidents(
 incident_id int primary key,
 incident_type varchar(100),
 incident_date date,
 location varchar(50),
 description text,
 status varchar(50),
 victim_id int,
 suspect_id int,
 foreign key (victim_id) references Victims(victim_id),
 foreign key (suspect_id) references Suspects(suspect_id)
);
```

```
insert into Incidents (incident_id,incident_type,incident_date,location,description,status,victim_id,
suspect_id)
```

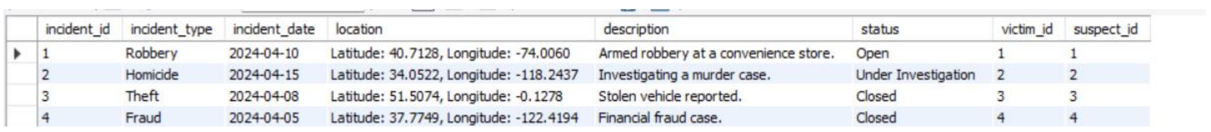
VALUES

(1, 'Robbery', '2024-04-10', 'Latitude: 40.7128, Longitude: -74.0060', 'Armed robbery at a convenience store.', 'Open', 1, 1),

(2, 'Homicide', '2024-04-15', 'Latitude: 34.0522, Longitude: -118.2437', 'Investigating a murder case.', 'Under Investigation', 2, 2),

(3, 'Theft', '2024-04-08', 'Latitude: 51.5074, Longitude: -0.1278', 'Stolen vehicle reported.', 'Closed', 3, 3),

(4, 'Fraud', '2024-04-05', 'Latitude: 37.7749, Longitude: -122.4194', 'Financial fraud case.', 'Closed', 4, 4);



incident_id	incident_type	incident_date	location	description	status	victim_id	suspect_id
1	Robbery	2024-04-10	Latitude: 40.7128, Longitude: -74.0060	Armed robbery at a convenience store.	Open	1	1
2	Homicide	2024-04-15	Latitude: 34.0522, Longitude: -118.2437	Investigating a murder case.	Under Investigation	2	2
3	Theft	2024-04-08	Latitude: 51.5074, Longitude: -0.1278	Stolen vehicle reported.	Closed	3	3
4	Fraud	2024-04-05	Latitude: 37.7749, Longitude: -122.4194	Financial fraud case.	Closed	4	4

4. Law Enforcement Agencies:

• AgencyID (Primary Key)

• AgencyName

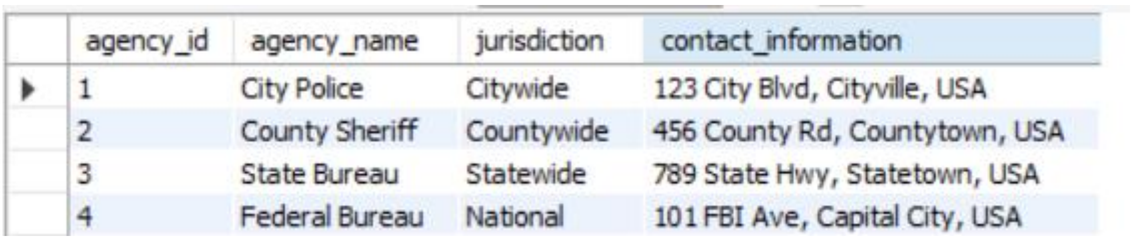
• Jurisdiction

• Contact Information

• Officer(s) (Link to Officers within the agency)

```
Create table LawEnforcementAgencies(

agency_id int primary key,

agency_name varchar(20),

jurisdiction varchar(50),

contact_information text

);


insert into LawEnforcementAgencies (agency_id, agency_name, jurisdiction, contact_information)

VALUES

(1,'City Police','Citywide', '123 City Blvd, Cityville, USA'),

(2, 'County Sheriff','Countywide', '456 County Rd, Countytown, USA'),

(3, 'State Bureau','Statewide', '789 State Hwy, Statetown, USA'),
```

(4, 'Federal Bureau','National', '101 FBI Ave, Capital City, USA');



	agency_id	agency_name	jurisdiction	contact_information
▶	1	City Police	Citywide	123 City Blvd, Cityville, USA
	2	County Sheriff	Countywide	456 County Rd, Countytown, USA
	3	State Bureau	Statewide	789 State Hwy, Statetown, USA
	4	Federal Bureau	National	101 FBI Ave, Capital City, USA

5. Officers:

• OfficerID (Primary Key)

• FirstName

• LastName

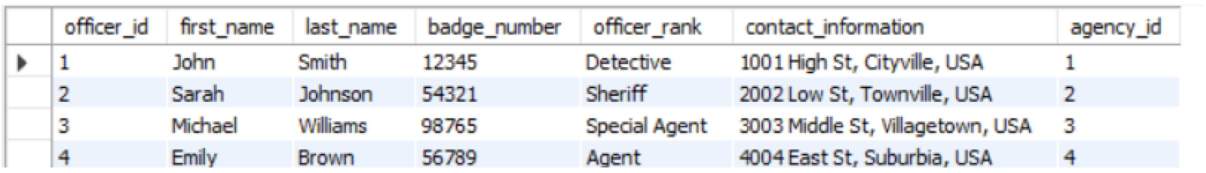
• BadgeNumber

• Rank

• Contact Information

• AgencyID (Foreign Key, linking to Law Enforcement Agencies)

```
Create table Officers(
officer_id int primary key,
first_name varchar(15),
last_name varchar(15),
badge_number varchar(20),
officer_rank varchar(20),
contact_information text,
agency_id int,
foreign key(agency_id) references LawEnforcementAgencies(agency_id)
);

INSERT INTO Officers (officer_id, first_name, last_name, badge_number, officer_rank,
contact_information, agency_id)
VALUES
(1, 'John', 'Smith', '12345', 'Detective', '1001 High St, Cityville, USA', 1),
(2, 'Sarah', 'Johnson', '54321', 'Sheriff', '2002 Low St, Townville, USA', 2),
(3, 'Michael', 'Williams', '98765', 'Special Agent', '3003 Middle St, Villagetown, USA', 3),
(4, 'Emily', 'Brown', '56789', 'Agent', '4004 East St, Suburbia, USA', 4);
```



officer_id	first_name	last_name	badge_number	officer_rank	contact_information	agency_id
1	John	Smith	12345	Detective	1001 High St, Cityville, USA	1
2	Sarah	Johnson	54321	Sheriff	2002 Low St, Townville, USA	2
3	Michael	Williams	98765	Special Agent	3003 Middle St, Villagetown, USA	3
4	Emily	Brown	56789	Agent	4004 East St, Suburbia, USA	4

6. Evidence:

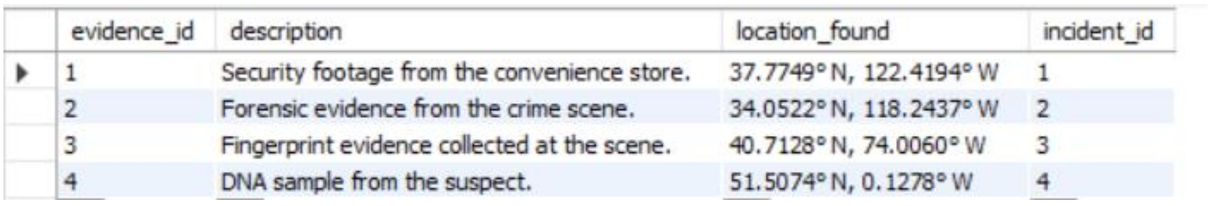
• EvidenceID (Primary Key)

• Description

• Location Found

• IncidentID (Foreign Key, linking to Incidents)

```
Create table Evidence(
evidence_id int primary key,
description text,
location_found varchar(50),
incident_id int,
foreign key(incident_id) references Incidents(incident_id)
);

INSERT INTO Evidence (evidence_id, description, location_found,incident_id)
VALUES
(1, 'Security footage from the convenience store.', '37.7749° N, 122.4194° W', 1),
(2, 'Forensic evidence from the crime scene.', '34.0522° N, 118.2437° W', 2),
(3, 'Fingerprint evidence collected at the scene.', '40.7128° N, 74.0060° W', 3),
(4, 'DNA sample from the suspect.', '51.5074° N, 0.1278° W', 4);
```



	evidence_id	description	location_found	incident_id
▶	1	Security footage from the convenience store.	37.7749° N, 122.4194° W	1
	2	Forensic evidence from the crime scene.	34.0522° N, 118.2437° W	2
	3	Fingerprint evidence collected at the scene.	40.7128° N, 74.0060° W	3
	4	DNA sample from the suspect.	51.5074° N, 0.1278° W	4

7. Reports:

• ReportID (Primary Key)

• IncidentID (Foreign Key, linking to Incidents)

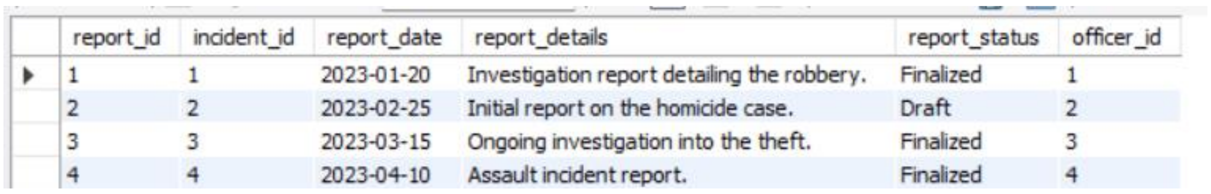
• ReportingOfficer (Foreign Key, linking to Officers)

• ReportDate

• ReportDetails

• Status (e.g., Draft, Finalized)

```
Create table Reports(
report_id int primary key,
incident_id int,
report_date date,
report_details text,
report_status varchar(50),
officer_id int,
foreign key(incident_id) references Incidents(incident_id),
foreign key(officer_id) references Officers(officer_id)
);
INSERT INTO Reports (report_id, incident_id, report_date, report_details,report_status,officer_id)
VALUES
(1, 1, '2023-01-20', 'Investigation report detailing the robbery.', 'Finalized',1),
(2, 2, '2023-02-25', 'Initial report on the homicide case.', 'Draft',2),
(3, 3, '2023-03-15', 'Ongoing investigation into the theft.', 'Finalized',3),
(4, 4, '2023-04-10', 'Assault incident report.', 'Finalized',4);
```



	report_id	incident_id	report_date	report_details	report_status	officer_id
▶	1	1	2023-01-20	Investigation report detailing the robbery.	Finalized	1
	2	2	2023-02-25	Initial report on the homicide case.	Draft	2
	3	3	2023-03-15	Ongoing investigation into the theft.	Finalized	3
	4	4	2023-04-10	Assault incident report.	Finalized	4

Coding

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters)

Service Provider Interface/Abstract class

• Keep the interfaces and implementation classes in package dao

Create ICrimeAnalysisService Interface/abstract classs with the following methods

```
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

    # Getters
    def get_evidence_id(self):
        return self.__evidence_id

    def get_description(self):
        return self.__description

    def get_location_found(self):
        return self.__location_found

    def get_incident_id(self):
        return self.__incident_id

    # Setters
    def set_evidence_id(self, evidence_id):
        self.__evidence_id = evidence_id

    def set_description(self, description):
        self.__description = description

    def set_location_found(self, location_found):
        self.__location_found = location_found

    def set_incident_id(self, incident_id):
        self.__incident_id = incident_id



class Incidents:
    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

    # Getters
    def get_incident_id(self):
        return self.__incident_id

    def get_incident_type(self):
        return self.__incident_type

    def get_incident_date(self):
        return self.__incident_date

    def get_location(self):
        return self.__location

    def get_description(self):
        return self.__description

    def get_status(self):
        return self.__status

    def get_victim_id(self):
        return self.__victim_id

    def get_suspect_id(self):
        return self.__suspect_id

    # Setters
    def set_incident_id(self, incident_id):
        self.__incident_id = incident_id

    def set_incident_type(self, incident_type):
        self.__incident_type = incident_type

    def set_incident_date(self, incident_date):
        self.__incident_date = incident_date

    def set_location(self, location):
        self.__location = location

    def set_description(self, description):
        self.__description = description

    def set_status(self, status):
        self.__status = status

    def set_victim_id(self, victim_id):
        self.__victim_id = victim_id

    def set_suspect_id(self, suspect_id):
        self.__suspect_id = suspect_id
```

```python
class LawEnforcementAgencies:
    def __init__(self, agency_id, agency_name, jurisdiction,
contact_information):
        self.__agency_id = agency_id
        self.__agency_name = agency_name
        self.__jurisdiction = jurisdiction
        self.__contact_information = contact_information

    # Getters
    def get_agency_id(self):
        return self.__agency_id

    def get_agency_name(self):
        return self.__agency_name

    def get_jurisdiction(self):
        return self.__jurisdiction

    def get_contact_information(self):
        return self.__contact_information

    # Setters
    def set_agency_id(self, agency_id):
        self.__agency_id = agency_id

    def set_agency_name(self, agency_name):
        self.__agency_name = agency_name

    def set_jurisdiction(self, jurisdiction):
        self.__jurisdiction = jurisdiction

    def set_contact_information(self, contact_information):
        self.__contact_information = contact_information
```

```python
class Officers:
    def __init__(self, officer_id, first_name, last_name, badge_number,
rank, contact_information, 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_information = contact_information
        self.__agency_id = agency_id

    # Getters
    def get_officer_id(self):
        return self.__officer_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_badge_number(self):
        return self.__badge_number
    def get_rank(self):
        return self.__rank

    def get_contact_information(self):
        return self.__contact_information

    def get_agency_id(self):
        return self.__agency_id

    # Setters
    def set_officer_id(self, officer_id):
        self.__officer_id = officer_id

    def set_first_name(self, first_name):
        self.__first_name = first_name

    def set_last_name(self, last_name):
        self.__last_name = last_name

    def set_badge_number(self, badge_number):
        self.__badge_number = badge_number

    def set_rank(self, rank):
        self.__rank = rank

    def set_contact_information(self, contact_information):
        self.__contact_information = contact_information

    def set_agency_id(self, agency_id):
        self.__agency_id = agency_id
```

```
class Reports:
    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

    # Getters
    def get_report_id(self):
        return self.__report_id

    def get_incident_id(self):
        return self.__incident_id

    def get_reporting_officer(self):
        return self.__reporting_officer

    def get_report_date(self):
        return self.__report_date

    def get_report_details(self):
        return self.__report_details
    def get_status(self):
        return self.__status

    # Setters
    def set_report_id(self, report_id):
        self.__report_id = report_id

    def set_incident_id(self, incident_id):
        self.__incident_id = incident_id

    def set_reporting_officer(self, reporting_officer):
        self.__reporting_officer = reporting_officer

    def set_report_date(self, report_date):
        self.__report_date = report_date

    def set_report_details(self, report_details):
        self.__report_details = report_details

    def set_status(self, status):
        self.__status = status
```

```
class Suspects:
    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

    # Getters
    def get_suspect_id(self):
        return self.__suspect_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_date_of_birth(self):
        return self.__date_of_birth

    def get_gender(self):
        return self.__gender

    def get_contact_information(self):
        return self.__contact_information

    # Setters
    def set_suspect_id(self, suspect_id):
        self.__suspect_id = suspect_id
    def set_first_name(self, first_name):
        self.__first_name = first_name

    def set_last_name(self, last_name):
        self.__last_name = last_name

    def set_date_of_birth(self, date_of_birth):
        self.__date_of_birth = date_of_birth

    def set_gender(self, gender):
        self.__gender = gender

    def set_contact_information(self, contact_information):
        self.__contact_information = contact_information
```

```
class Victims:
    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

    # Getters
    def get_victim_id(self):
        return self.__victim_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_date_of_birth(self):
        return self.__date_of_birth

    def get_gender(self):
        return self.__gender

    def get_contact_information(self):
        return self.__contact_information

    # Setters
    def set_victim_id(self, victim_id):
        self.__victim_id = victim_id

    def set_first_name(self, first_name):
        self.__first_name = first_name

    def set_last_name(self, last_name):
        self.__last_name = last_name

    def set_date_of_birth(self, date_of_birth):
        self.__date_of_birth = date_of_birth
```

```python
    def set_gender(self, gender):
        self.__gender = gender

    def set_contact_information(self, contact_information):
        self.__contact_information = contact_information
```

```python
class ICase:
    def __init__(self, case_id, case_type, case_date, status, officer_id):
        self.__case_id = case_id
        self.__case_type = case_type
        self.__case_date = case_date
        self.__status = status
        self.__officer_id = officer_id

    # Getters
    def get_case_id(self):
        return self.__case_id

    def get_case_type(self):
        return self.__case_type

    def get_case_date(self):
        return self.__case_date

    def get_status(self):
        return self.__status

    def get_officer_id(self):
        return self.__officer_id

    # Setters
    def set_case_id(self, case_id):
        self.__case_id = case_id

    def set_case_type(self, case_type):
        self.__case_type = case_type

    def set_case_date(self, case_date):
        self.__case_date = case_date

    def set_status(self, status):
        self.__status = status

    def set_officer_id(self, officer_id):
        self.__officer_id = officer_id
```

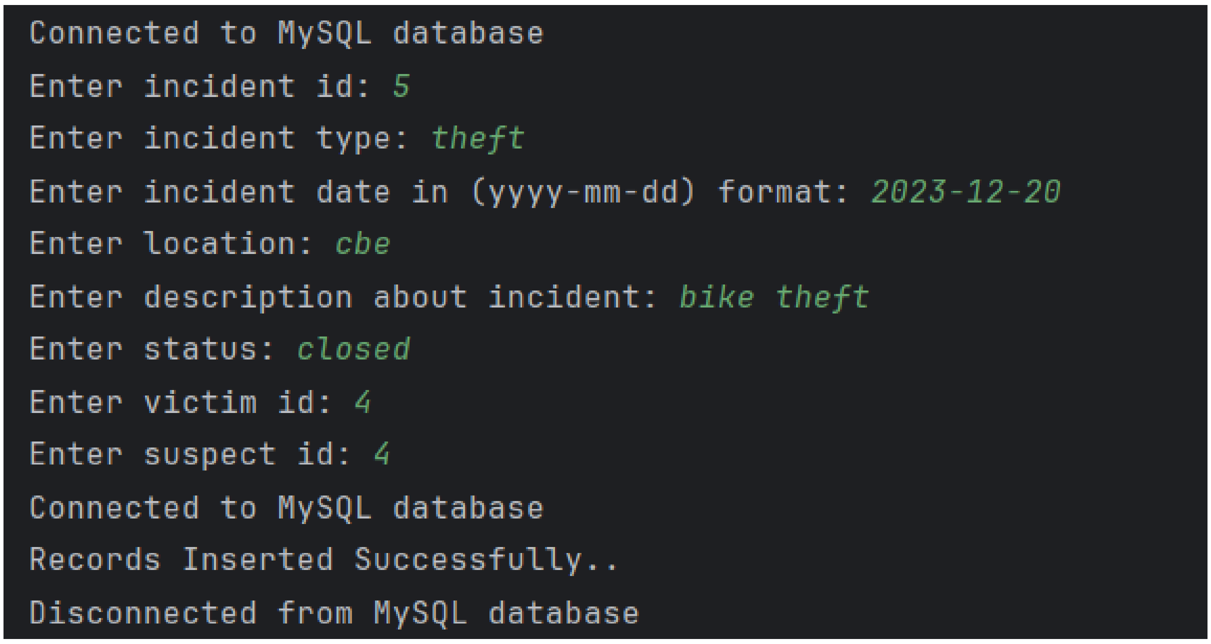
// Create a new incident

createIncident();

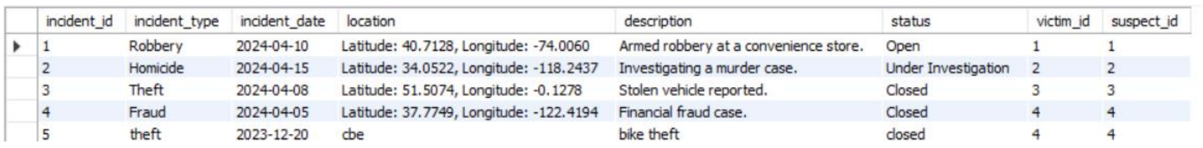
parameters- Incident object

return type Boolean

```
from util.DBConnUtil import dbConnection
from entity.Incidents import Incidents
from entity.ICase import ICase
from MyExceptions.InvalidNameError import InvalidNameError, StringCheck
from MyExceptions.IncidentNumberNotFoundException import
IncidentNumberNotFoundException

class ICrimeAnalysisService(dbConnection):
    def __init__(self, host, username, password, port, database):
        super().__init__(host, username, password, port, database)
        self.incident_id = ""
        self.incident_type = ""
        self.incident_date = ""
        self.location = ""
        self.description = ""
        self.status = ""
        self.victim_id = ""
        self.suspect_id = ""

    def create_incident(self):
        try:
            incident_id = input("Enter incident id: ")
            self.incident_id = incident_id

            incident_type = input("Enter incident type: ")
            self.incident_type = incident_type

            incident_date = input("Enter incident date in (yyyy-mm-dd)
format: ")
            self.incident_date = incident_date

            location = input("Enter location: ")
            self.location = location

            description = input("Enter description about incident: ")
            self.description = description

            status = input("Enter status: ")
            self.status = status

            victim_id = input("Enter victim id: ")
            self.victim_id = victim_id

            suspect_id = input("Enter suspect id: ")
            self.suspect_id = suspect_id

            # SQL query to insert a new incident
            sql_query = """
                        INSERT INTO Incidents (incident_id, incident_type,
incident_date, location, description, status, victim_id, suspect_id)
                        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                        """
            # Parameters for the SQL query
            values = [(self.incident_id, self.incident_type,
self.incident_date,
                      self.location, self.description, self.status,
self.victim_id, self.suspect_id)]
            self.open()
            self.stmt.executemany(sql_query, values)
            self.connection.commit()
            print('Records Inserted Successfully..')
            self.close()
            return True  # Return True if the operation is successful
        except Exception as e:
            print(f"Error creating incident: {e}")
            return False  # Return False if the operation fails
```



```
Connected to MySQL database
Enter incident id: 5
Enter incident type: theft
Enter incident date in (yyyy-mm-dd) format: 2023-12-20
Enter location: cbe
Enter description about incident: bike theft
Enter status: closed
Enter victim id: 4
Enter suspect id: 4
Connected to MySQL database
Records Inserted Successfully..
Disconnected from MySQL database
```



incident_id	incident_type	incident_date	location	description	status	victim_id	suspect_id
1	Robbery	2024-04-10	Latitude: 40.7128, Longitude: -74.0060	Armed robbery at a convenience store.	Open	1	1
2	Homicide	2024-04-15	Latitude: 34.0522, Longitude: -118.2437	Investigating a murder case.	Under Investigation	2	2
3	Theft	2024-04-08	Latitude: 51.5074, Longitude: -0.1278	Stolen vehicle reported.	Closed	3	3
4	Fraud	2024-04-05	Latitude: 37.7749, Longitude: -122.4194	Financial fraud case.	Closed	4	4
5	theft	2023-12-20	cbe	bike theft	closed	4	4

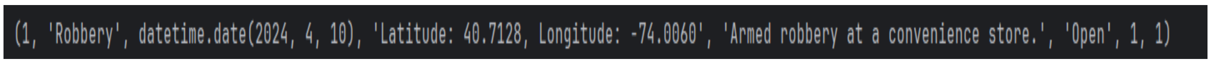
// Update the status of an incident

updateIncidentStatus();

parameters- Status object,incidentid

return type Boolean

```
def update_incident_status(self, incident_id, status):
    try:
        # Check if the incident ID exists in the database
        check_query = """SELECT incident_id FROM Incidents WHERE
incident_id = %s"""
        self.open()
        self.stmt.execute(check_query, (incident_id,))
        record = self.stmt.fetchone()

        if record:
            # Update the status of the incident
            sql_query = """UPDATE Incidents SET Status = %s WHERE
incident_id = %s"""
            self.stmt.execute(sql_query, (status, incident_id))
            self.conn.commit()
            print('Record Updated Successfully')
            return True
        else:
            raise IncidentNumberNotFoundException("Incident ID not found")

    except Exception as e:
        print(f"Error updating incident status: {e}")
        return False
```



```
(1, 'Robbery', datetime.date(2024, 4, 10), 'Latitude: 40.7128, Longitude: -74.0060', 'Armed robbery at a convenience store.', 'Open', 1, 1)
```

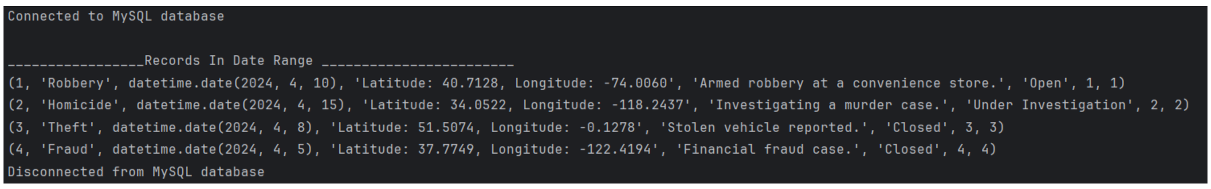
// Get a list of incidents within a date range

getIncidentsInDateRange();

parameters- startDate, endDate

return type Collection of Incident objects

```
def get_incidents_in_date_range(self, start_date, end_date):

    # SQL query to select incidents within the specified date range
    sql_query = """ SELECT * FROM Incidents
                    WHERE incident_date BETWEEN %s AND %s
                """
    # Parameters for the SQL query
    values = (start_date, end_date)
    self.open()
    self.stmt.execute(sql_query, values)
    recods = self.stmt.fetchall()
    print('')
    print('_________________Records In Date Range
________________________')
    for i in recods:
        print(i)
    self.close()
    if not recods:
        print("No Incidents Found in the Given Dates")
    return f'Incidents details fetched successfully'
```



```
Connected to MySQL database

_________________Records In Date Range ________________________
(1, 'Robbery', datetime.date(2024, 4, 10), 'Latitude: 40.7128, Longitude: -74.0060', 'Armed robbery at a convenience store.', 'Open', 1, 1)
(2, 'Homicide', datetime.date(2024, 4, 15), 'Latitude: 34.0522, Longitude: -118.2437', 'Investigating a murder case.', 'Under Investigation', 2, 2)
(3, 'Theft', datetime.date(2024, 4, 8), 'Latitude: 51.5074, Longitude: -0.1278', 'Stolen vehicle reported.', 'Closed', 3, 3)
(4, 'Fraud', datetime.date(2024, 4, 5), 'Latitude: 37.7749, Longitude: -122.4194', 'Financial fraud case.', 'Closed', 4, 4)
Disconnected from MySQL database
```

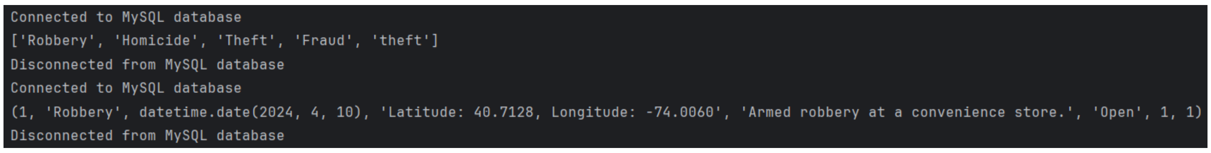
// Search for incidents based on various criteria

searchIncidents(IncidentType criteria);

parameters- IncidentType object

return type Collection of Incident objects

```
def search_incidents(self, incident_type):
    try:
        check_query = """ SELECT incident_type FROM Incidents"""
        self.open()
        self.stmt.execute(check_query)
        records = self.stmt.fetchall()
        checking = [i[0] for i in records]
        print(checking)
        self.close()
        if not isinstance(incident_type, str):
            print("invalid data type")
        incident_type = incident_type.lower()
        for incident_type in checking:
            incident_type = incident_type.lower()
            if incident_type == incident_type:
                # SQL query to search for incidents based on the given
criteria
                sql_query = """
                               SELECT *
                               FROM Incidents
                               WHERE incident_type = %s
                            """
                # Parameters for the SQL query
                values = [(incident_type)]
                self.open()
                self.stmt.execute(sql_query, values)
                recods = self.stmt.fetchall()
                for i in recods:
                    print(i)
                self.close()
            else:
                print(f"No Incident found with Incident type:
{incident_type}")

            return f'Incidents details fetched successfully'
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```



```
Connected to MySQL database
['Robbery', 'Homicide', 'Theft', 'Fraud', 'theft']
Disconnected from MySQL database
Connected to MySQL database
(1, 'Robbery', datetime.date(2024, 4, 10), 'Latitude: 40.7128, Longitude: -74.0060', 'Armed robbery at a convenience store.', 'Open', 1, 1)
Disconnected from MySQL database
```

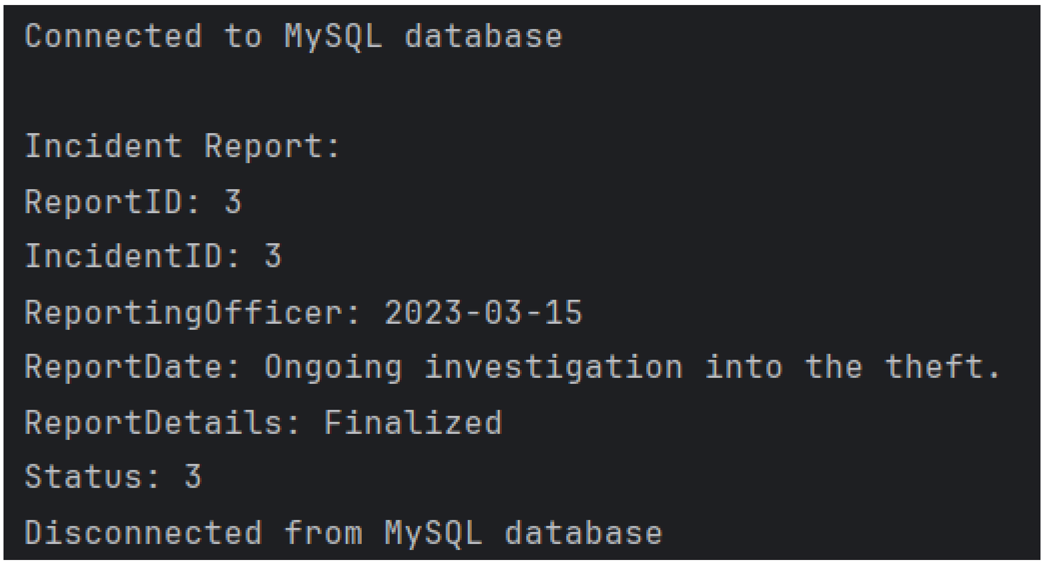
// Generate incident reports

generateIncidentReport();

parameters- Incident object

return type Report object

```
def generate_incident_report(self, incident_id):
    try:
        self.open()
        select_Incidents_str = '''
                            SELECT * FROM Reports
                            WHERE incident_id = %s
                        '''
        self.stmt.execute(select_Incidents_str, (incident_id,))
        customer_data = self.stmt.fetchone()

        if not customer_data:
            raise IncidentNumberNotFoundException()
        else:
            print('\nIncident Report:')
            print(f"ReportID: {customer_data[0]}")
            print(f"IncidentID: {customer_data[1]}")
            print(f"ReportingOfficer: {customer_data[2]}")
            print(f"ReportDate: {customer_data[3]}")
            print(f"ReportDetails: {customer_data[4]}")
            print(f"Status: {customer_data[5]}")
            # print(f"VictimID: {customer_data[6]}")
            # print(f"SuspectID: {customer_data[7]}")

        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```



```
Connected to MySQL database

Incident Report:
ReportID: 3
IncidentID: 3
ReportingOfficer: 2023-03-15
ReportDate: Ongoing investigation into the theft.
ReportDetails: Finalized
Status: 3
Disconnected from MySQL database
```

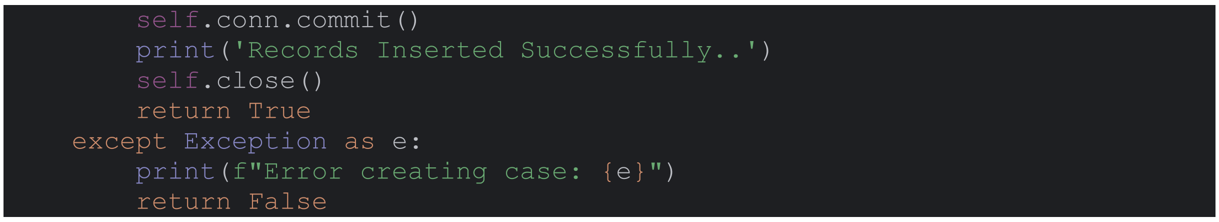
// Create a new case and associate it with incidents

createCase();

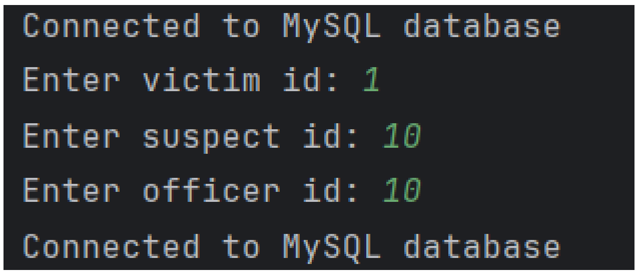
parameters- caseDescription string, collection of Incident Objects

return type Case object

```
def create_case(self, incident_id):
    try:
        victim_id = int(input("Enter victim id: "))
        suspect_id = int(input("Enter suspect id: "))
        officer_id = int(input("Enter officer id: "))

        query = """INSERT INTO ICase(incident_id, incident_type,
victim_name, suspect_name, officer_id, case_description)
                    SELECT
                        i.incident_id,
                        i.incident_type,
                        CONCAT(v.FirstName, ' ', v.LastName) AS
victim_name,
                        CONCAT(s.FirstName, ' ', s.LastName) AS
suspect_name,
                        %s AS officer_id,
                        i.description AS case_description
                    FROM
                        Incidents i
                        LEFT JOIN Victims v ON i.victim_id = v.victim_id
                        LEFT JOIN Suspects s ON i.suspect_id = s.suspect_id
                    WHERE
                        i.victim_id = %s
                        AND i.suspect_id = %s
                        AND i.incident_id = %s
                """
        values = (officer_id, victim_id, suspect_id, incident_id)
        self.open()
        self.stmt.execute(query, values)
```



```
        self.conn.commit()
        print('Records Inserted Successfully..')
        self.close()
        return True
    except Exception as e:
        print(f"Error creating case: {e}")
        return False
```



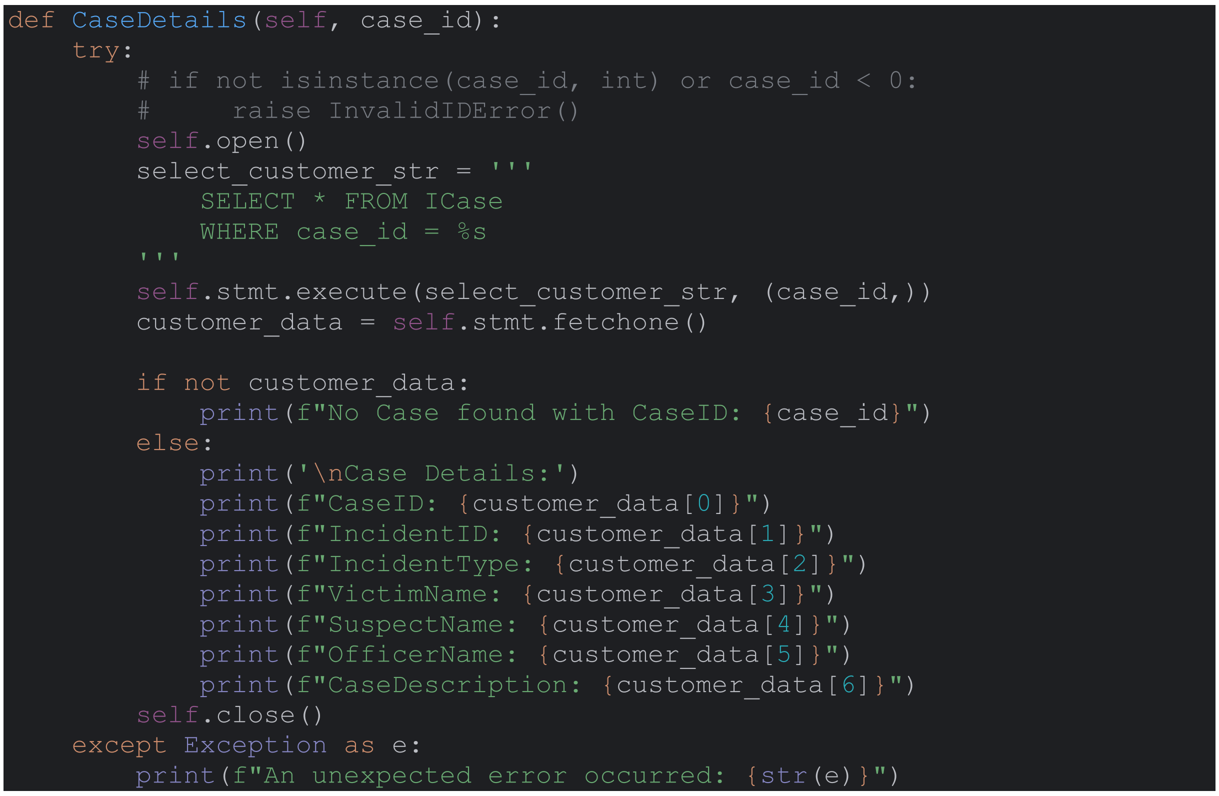
```
Connected to MySQL database
Enter victim id: 1
Enter suspect id: 10
Enter officer id: 10
Connected to MySQL database
```

// Get details of a specific case

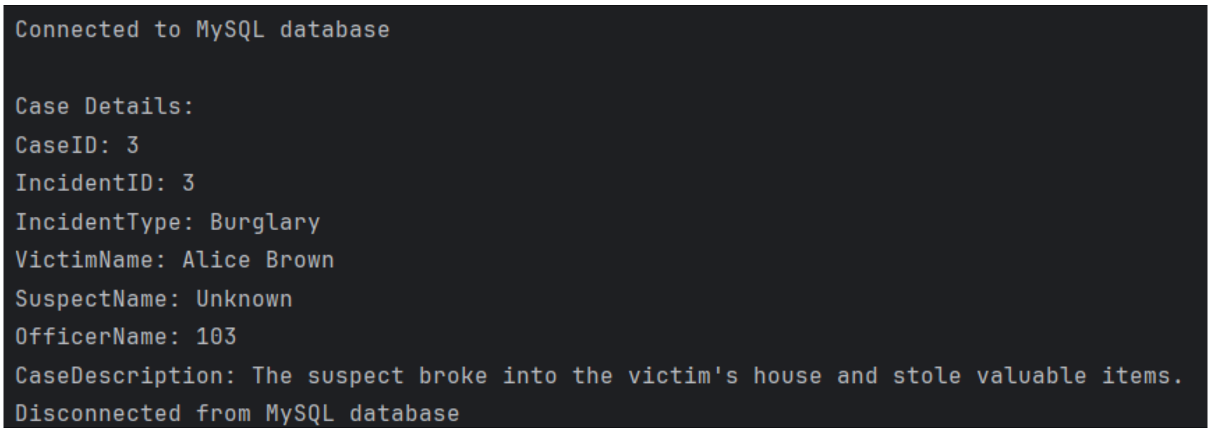
Case getCaseDetails(int caseId);

parameters- caseDescription string, collection of Incident Objects

return type Case object



```
def CaseDetails(self, case_id):
    try:
        # if not isinstance(case_id, int) or case_id < 0:
        #     raise InvalidIDError()
        self.open()
        select_customer_str = '''
            SELECT * FROM ICase
            WHERE case_id = %s
        '''
        self.stmt.execute(select_customer_str, (case_id,))
        customer_data = self.stmt.fetchone()

        if not customer_data:
            print(f"No Case found with CaseID: {case_id}")
        else:
            print('\nCase Details:')
            print(f"CaseID: {customer_data[0]}")
            print(f"IncidentID: {customer_data[1]}")
            print(f"IncidentType: {customer_data[2]}")
            print(f"VictimName: {customer_data[3]}")
            print(f"SuspectName: {customer_data[4]}")
            print(f"OfficerName: {customer_data[5]}")
            print(f"CaseDescription: {customer_data[6]}")
        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```



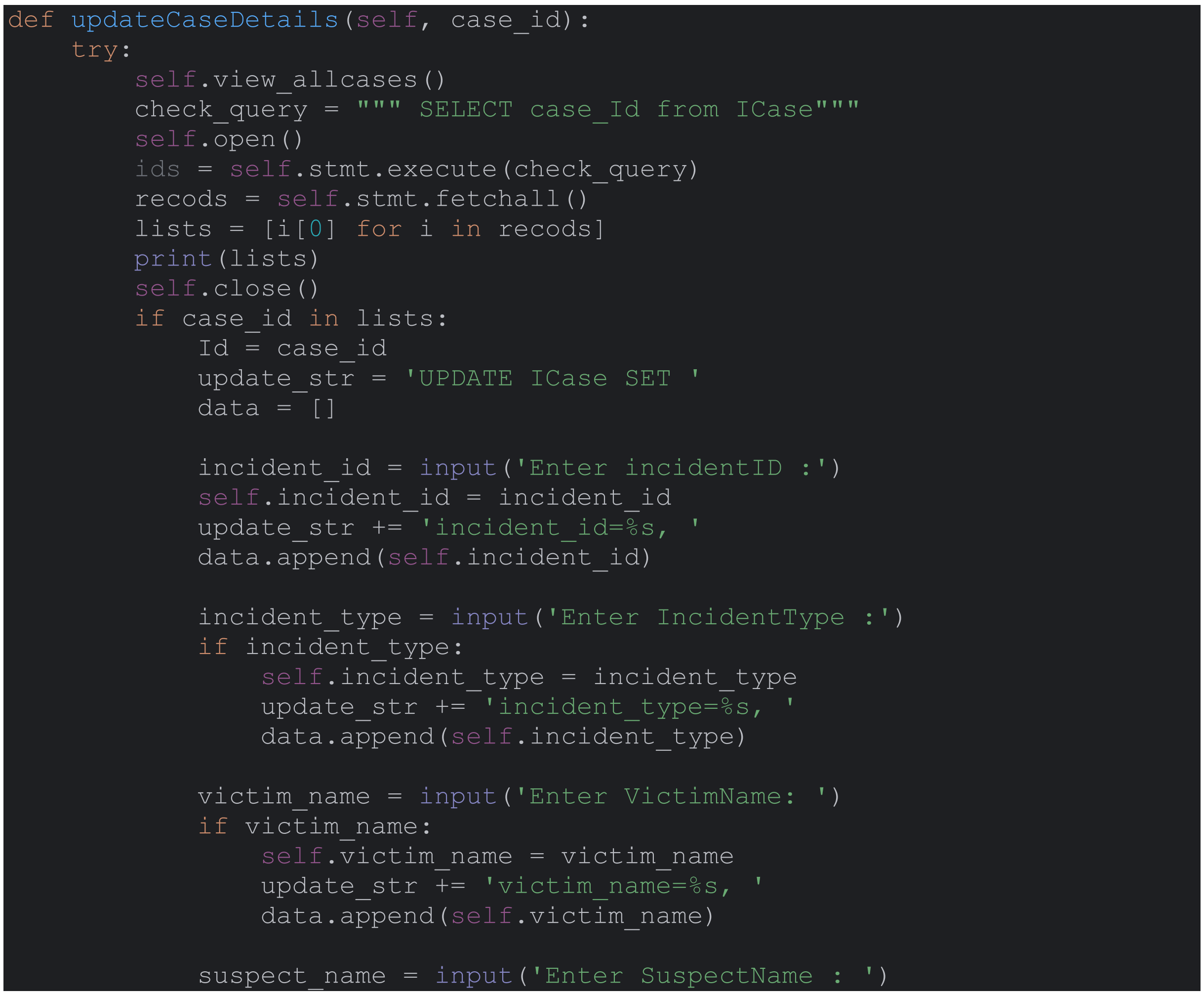
```
Connected to MySQL database

Case Details:
CaseID: 3
IncidentID: 3
IncidentType: Burglary
VictimName: Alice Brown
SuspectName: Unknown
OfficerName: 103
CaseDescription: The suspect broke into the victim's house and stole valuable items.
Disconnected from MySQL database
```

// Update case details

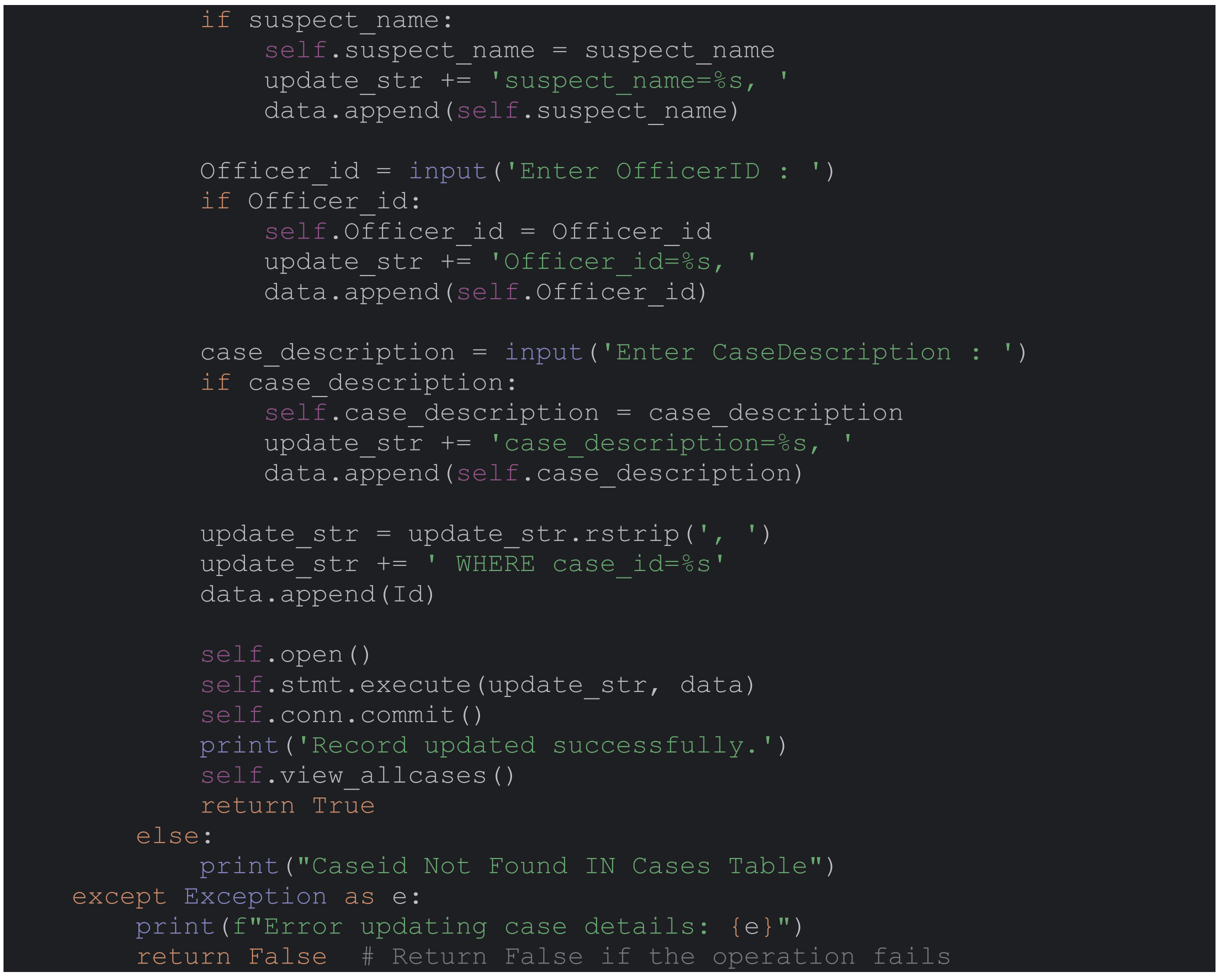
updateCaseDetails();

parameters- Case object

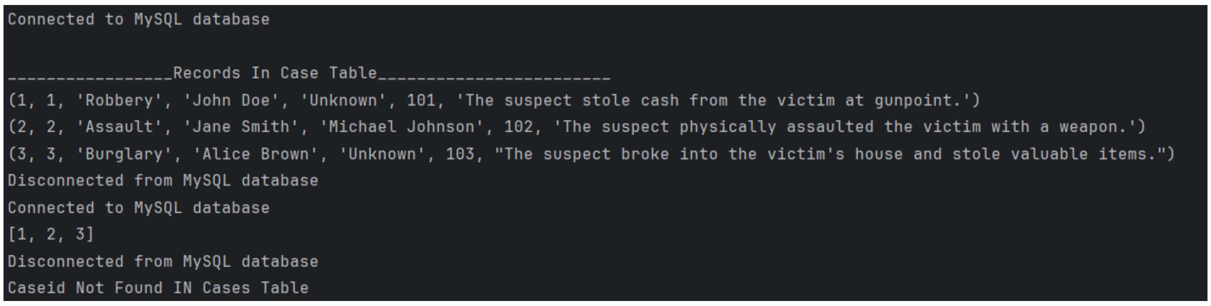
return type Boolean



```
def updateCaseDetails(self, case_id):
    try:
        self.view_allcases()
        check_query = """ SELECT case_Id from ICase"""
        self.open()
        ids = self.stmt.execute(check_query)
        recods = self.stmt.fetchall()
        lists = [i[0] for i in recods]
        print(lists)
        self.close()
        if case_id in lists:
            Id = case_id
            update_str = 'UPDATE ICase SET '
            data = []

            incident_id = input('Enter incidentID :')
            self.incident_id = incident_id
            update_str += 'incident_id=%s, '
            data.append(self.incident_id)

            incident_type = input('Enter IncidentType :')
            if incident_type:
                self.incident_type = incident_type
                update_str += 'incident_type=%s, '
                data.append(self.incident_type)

            victim_name = input('Enter VictimName: ')
            if victim_name:
                self.victim_name = victim_name
                update_str += 'victim_name=%s, '
                data.append(self.victim_name)

            suspect_name = input('Enter SuspectName : ')
```



```
            if suspect_name:
                self.suspect_name = suspect_name
                update_str += 'suspect_name=%s, '
                data.append(self.suspect_name)

            Officer_id = input('Enter OfficerID : ')
            if Officer_id:
                self.Officer_id = Officer_id
                update_str += 'Officer_id=%s, '
                data.append(self.Officer_id)

            case_description = input('Enter CaseDescription : ')
            if case_description:
                self.case_description = case_description
                update_str += 'case_description=%s, '
                data.append(self.case_description)

            update_str = update_str.rstrip(', ')
            update_str += ' WHERE case_id=%s'
            data.append(Id)

            self.open()
            self.stmt.execute(update_str, data)
            self.conn.commit()
            print('Record updated successfully.')
            self.view_allcases()
            return True
        else:
            print("Caseid Not Found IN Cases Table")
    except Exception as e:
        print(f"Error updating case details: {e}")
        return False  # Return False if the operation fails
```



```
Connected to MySQL database

_________________Records In Case Table________________________
(1, 1, 'Robbery', 'John Doe', 'Unknown', 101, 'The suspect stole cash from the victim at gunpoint.')
(2, 2, 'Assault', 'Jane Smith', 'Michael Johnson', 102, 'The suspect physically assaulted the victim with a weapon.')
(3, 3, 'Burglary', 'Alice Brown', 'Unknown', 103, "The suspect broke into the victim's house and stole valuable items.")
Disconnected from MySQL database
Connected to MySQL database
[1, 2, 3]
Disconnected from MySQL database
Caseid Not Found IN Cases Table
```

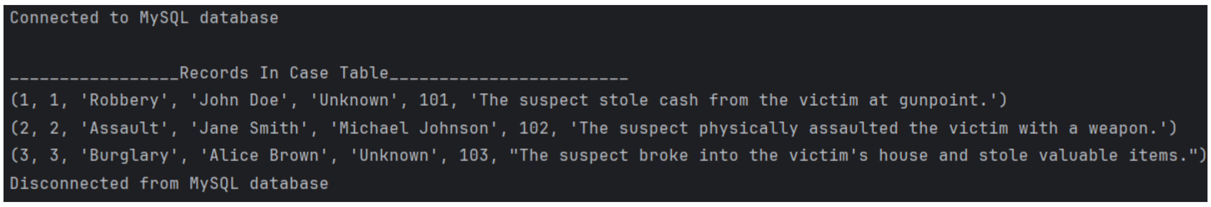
// Get a list of all cases

List<Case> getAllCases();

parameters- None

return type Collection of cases

```
def view_allcases(self):
    self.open()
    select_str = '''select * from Icase '''
    self.stmt.execute(select_str)
    recods = self.stmt.fetchall()
    print('')
    print('_________________Records In Case Table________________________')
    for i in recods:
        print(i)
    self.close()
    return f'Case details fetched successfully'
```



```
Connected to MySQL database

_________________Records In Case Table________________________
(1, 1, 'Robbery', 'John Doe', 'Unknown', 101, 'The suspect stole cash from the victim at gunpoint.')
(2, 2, 'Assault', 'Jane Smith', 'Michael Johnson', 102, 'The suspect physically assaulted the victim with a weapon.')
(3, 3, 'Burglary', 'Alice Brown', 'Unknown', 103, "The suspect broke into the victim's house and stole valuable items.")
Disconnected from MySQL database
```

7: Connect your application to the SQL database:

1. Write code to establish a connection to your SQL database.

Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.

Connection properties supplied in the connection string should be read from a property file.

Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.

```
import mysql.connector as connection

from util.PropertyUtil import PropertyUtil

class dbConnection():

    def _init_(self):

        pass

    def open(self):

        try:
            l = PropertyUtil.getPropertyString(self)
            self.conn = connection.connect(host=l[0], database=l[3],
username=l[1], password=l[2])
            if self.conn:
                print("--Database Is Connected--")
            self.stmt = self.conn.cursor()
        except Exception as e:
            print(e)


    def close(self):
        try:
            self.conn.close()
            print('Connection Closed.')
        except Exception as e:
            print(e)


d = dbConnection()
d.open()
d.close()
```

```
class PropertyUtil:


    def getPropertyString(self):
        host = 'localhost'
        username = 'root'
        password = 'root'
        database = 'crime_reporting'
        return host, username, password, database
```

2. Provide implementation for all the methods in the interface/abstract clsass

```python
from util.DBConnUtil import dbConnection
from entity.Incidents import Incidents
from entity.ICase import ICase
from MyExceptions.InvalidNameError import InvalidNameError, StringCheck
from MyExceptions.IncidentNumberNotFoundException import
IncidentNumberNotFoundException

class ICrimeAnalysisService(dbConnection):
    def __init__(self, host, username, password, port, database):
        super().__init__(host, username, password, port, database)
        self.incident_id = ""
        self.incident_type = ""
        self.incident_date = ""
        self.location = ""
        self.description = ""
        self.status = ""
        self.victim_id = ""
        self.suspect_id = ""

    def create_incident(self):
        try:
            incident_id = input("Enter incident id: ")
            self.incident_id = incident_id

            incident_type = input("Enter incident type: ")
            self.incident_type = incident_type

            incident_date = input("Enter incident date in (yyyy-mm-dd)
format: ")
            self.incident_date = incident_date

            location = input("Enter location: ")
            self.location = location

            description = input("Enter description about incident: ")
            self.description = description

            status = input("Enter status: ")
            self.status = status

            victim_id = input("Enter victim id: ")
            self.victim_id = victim_id

            suspect_id = input("Enter suspect id: ")
            self.suspect_id = suspect_id

            # SQL query to insert a new incident
            sql_query = """
                        INSERT INTO Incidents (incident_id, incident_type,
incident_date, location, description, status, victim_id, suspect_id)
                        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                        """
            # Parameters for the SQL query
            values = [(self.incident_id, self.incident_type,
self.incident_date,
                      self.location, self.description, self.status,
self.victim_id, self.suspect_id)]
            self.open()
            self.stmt.executemany(sql_query, values)
            self.connection.commit()
            print('Records Inserted Successfully..')
            self.close()
            return True  # Return True if the operation is successful
        except Exception as e:
            print(f"Error creating incident: {e}")
            return False  # Return False if the operation fails

    def update_incident_status(self, incident_id, status):
        try:
            # Check if the incident ID exists in the database
            check_query = """SELECT incident_id FROM Incidents WHERE
incident_id = %s"""
            self.open()
            self.stmt.execute(check_query, (incident_id,))
            record = self.stmt.fetchone()

            if record:
                # Update the status of the incident
                sql_query = """UPDATE Incidents SET Status = %s WHERE
incident_id = %s"""
                self.stmt.execute(sql_query, (status, incident_id))
                self.conn.commit()
                print('Record Updated Successfully')
                return True
            else:
                raise IncidentNumberNotFoundException("Incident ID not
found")

        except Exception as e:
            print(f"Error updating incident status: {e}")
            return False

    def get_incidents_in_date_range(self, start_date, end_date):

        # SQL query to select incidents within the specified date range
        sql_query = """ SELECT * FROM Incidents
                        WHERE incident_date BETWEEN %s AND %s
                    """
        # Parameters for the SQL query
        values = (start_date, end_date)
        self.open()
        self.stmt.execute(sql_query, values)
        recods = self.stmt.fetchall()
        print('')
        print('_________________Records In Date Range
________________________')
        for i in recods:
            print(i)
        self.close()

        if not recods:
            print("No Incidents Found in the Given Dates")
        return f'Incidents details fetched successfully'

    def search_incidents(self, incident_type):
        try:
            check_query = """ SELECT incident_type FROM Incidents"""
            self.open()
            self.stmt.execute(check_query)
            records = self.stmt.fetchall()
            checking = [i[0] for i in records]
            print(checking)
            self.close()
            if not isinstance(incident_type, str):
                print("invalid data type")
            incident_type = incident_type.lower()
            for incident_type in checking:
                incident_type = incident_type.lower()
                if incident_type == incident_type:
                    # SQL query to search for incidents based on the given
criteria
                    sql_query = """
                                   SELECT *
                                   FROM Incidents
                                   WHERE incident_type = %s
                                """
                    # Parameters for the SQL query
                    values = [(incident_type)]
                    self.open()
                    self.stmt.execute(sql_query, values)
                    recods = self.stmt.fetchall()
                    for i in recods:
                        print(i)
                    self.close()
                else:
                    print(f"No Incident found with Incident type:
{incident_type}")

                return f'Incidents details fetched successfully'
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def generate_incident_report(self, incident_id):
        try:
            self.open()
            select_Incidents_str = '''
                                SELECT * FROM Reports
                                WHERE incident_id = %s
                            '''
            self.stmt.execute(select_Incidents_str, (incident_id,))
            customer_data = self.stmt.fetchone()

            if not customer_data:
                raise IncidentNumberNotFoundException()
            else:
                print('\nIncident Report:')
                print(f"ReportID: {customer_data[0]}")
                print(f"IncidentID: {customer_data[1]}")
                print(f"ReportingOfficer: {customer_data[2]}")
                print(f"ReportDate: {customer_data[3]}")
                print(f"ReportDetails: {customer_data[4]}")
                print(f"Status: {customer_data[5]}")
                # print(f"VictimID: {customer_data[6]}")
                # print(f"SuspectID: {customer_data[7]}")

            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def create_case(self, incident_id):
        try:
            victim_id = int(input("Enter victim id: "))
            suspect_id = int(input("Enter suspect id: "))
            officer_id = int(input("Enter officer id: "))

            query = """INSERT INTO ICase(incident_id, incident_type,
victim_name, suspect_name, officer_id, case_description)
                        SELECT
                            i.incident_id,
                            i.incident_type,
                            CONCAT(v.FirstName, ' ', v.LastName) AS
victim_name,
                            CONCAT(s.FirstName, ' ', s.LastName) AS
suspect_name,
                            %s AS officer_id,
                            i.description AS case_description
                        FROM
                            Incidents i
                            LEFT JOIN Victims v ON i.victim_id =
v.victim_id
                            LEFT JOIN Suspects s ON i.suspect_id =
s.suspect_id
                        WHERE
                            i.victim_id = %s
                            AND i.suspect_id = %s
                            AND i.incident_id = %s
                    """
            values = (officer_id, victim_id, suspect_id, incident_id)
            self.open()
            self.stmt.execute(query, values)
            self.conn.commit()
            print('Records Inserted Successfully..')
            self.close()
            return True
        except Exception as e:
            print(f"Error creating case: {e}")
            return False

    def CaseDetails(self, case_id):
        try:
            # if not isinstance(case_id, int) or case_id < 0:
            #     raise InvalidIDError()
            self.open()
            select_customer_str = '''
                SELECT * FROM ICase
                WHERE case_id = %s
            '''
            self.stmt.execute(select_customer_str, (case_id,))
            customer_data = self.stmt.fetchone()

            if not customer_data:
                print(f"No Case found with CaseID: {case_id}")
            else:
                print('\nCase Details:')
                print(f"CaseID: {customer_data[0]}")
                print(f"IncidentID: {customer_data[1]}")
                print(f"IncidentType: {customer_data[2]}")
                print(f"VictimName: {customer_data[3]}")
```

```python
                print(f"SuspectName: {customer_data[4]}")
                print(f"OfficerName: {customer_data[5]}")
                print(f"CaseDescription: {customer_data[6]}")
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def updateCaseDetails(self, case_id):
        try:
            self.view_allcases()
            check_query = """ SELECT case_Id from ICase"""
            self.open()
            ids = self.stmt.execute(check_query)
            recods = self.stmt.fetchall()
            lists = [i[0] for i in recods]
            print(lists)
            self.close()
            if case_id in lists:
                Id = case_id
                update_str = 'UPDATE ICase SET '
                data = []

                incident_id = input('Enter incidentID :')
                self.incident_id = incident_id
                update_str += 'incident_id=%s, '
                data.append(self.incident_id)

                incident_type = input('Enter IncidentType :')
                if incident_type:
                    self.incident_type = incident_type
                    update_str += 'incident_type=%s, '
                    data.append(self.incident_type)

                victim_name = input('Enter VictimName: ')
                if victim_name:
                    self.victim_name = victim_name
                    update_str += 'victim_name=%s, '
                    data.append(self.victim_name)

                suspect_name = input('Enter SuspectName : ')
                if suspect_name:
                    self.suspect_name = suspect_name
                    update_str += 'suspect_name=%s, '
                    data.append(self.suspect_name)

                Officer_id = input('Enter OfficerID : ')
                if Officer_id:
                    self.Officer_id = Officer_id
                    update_str += 'Officer_id=%s, '
                    data.append(self.Officer_id)

                case_description = input('Enter CaseDescription : ')
                if case_description:
                    self.case_description = case_description
                    update_str += 'case_description=%s, '
                    data.append(self.case_description)

                update_str = update_str.rstrip(', ')
                update_str += ' WHERE case_id=%s'
                data.append(Id)
                self.open()
                self.stmt.execute(update_str, data)
                self.conn.commit()
                print('Record updated successfully.')
                self.view_allcases()
                return True
            else:
                print("Caseid Not Found IN Cases Table")
        except Exception as e:
            print(f"Error updating case details: {e}")
            return False  # Return False if the operation fails

    def view_allcases(self):
        self.open()
        select_str = '''select * from Icase '''
        self.stmt.execute(select_str)
        recods = self.stmt.fetchall()
        print('')
        print('_________________Records In Case
Table________________________')
        for i in recods:
            print(i)
        self.close()
        return f'Case details fetched successfully'
```

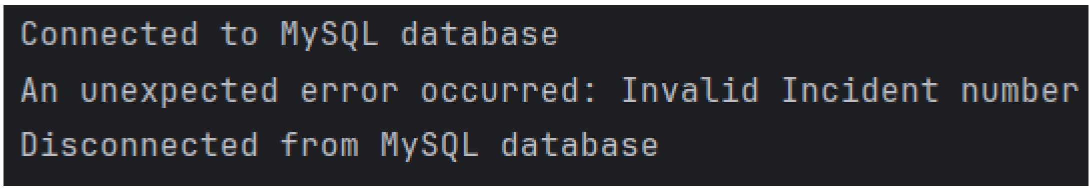
8: Exception Handling

Create the exceptions in package c.myexceptions

Define the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,

1. IncidentNumberNotFoundException :throw this exception when user enters an invalid patient number which doesn’t exist in db

```
class  IncidentNumberNotFoundException(Exception):
    def __init__(self, message="Invalid Incident number "):
        self.message = message
        super().__init__(self.message)
```



```
Connected to MySQL database
An unexpected error occurred: Invalid Incident number
Disconnected from MySQL database
```
]

9. Main Method

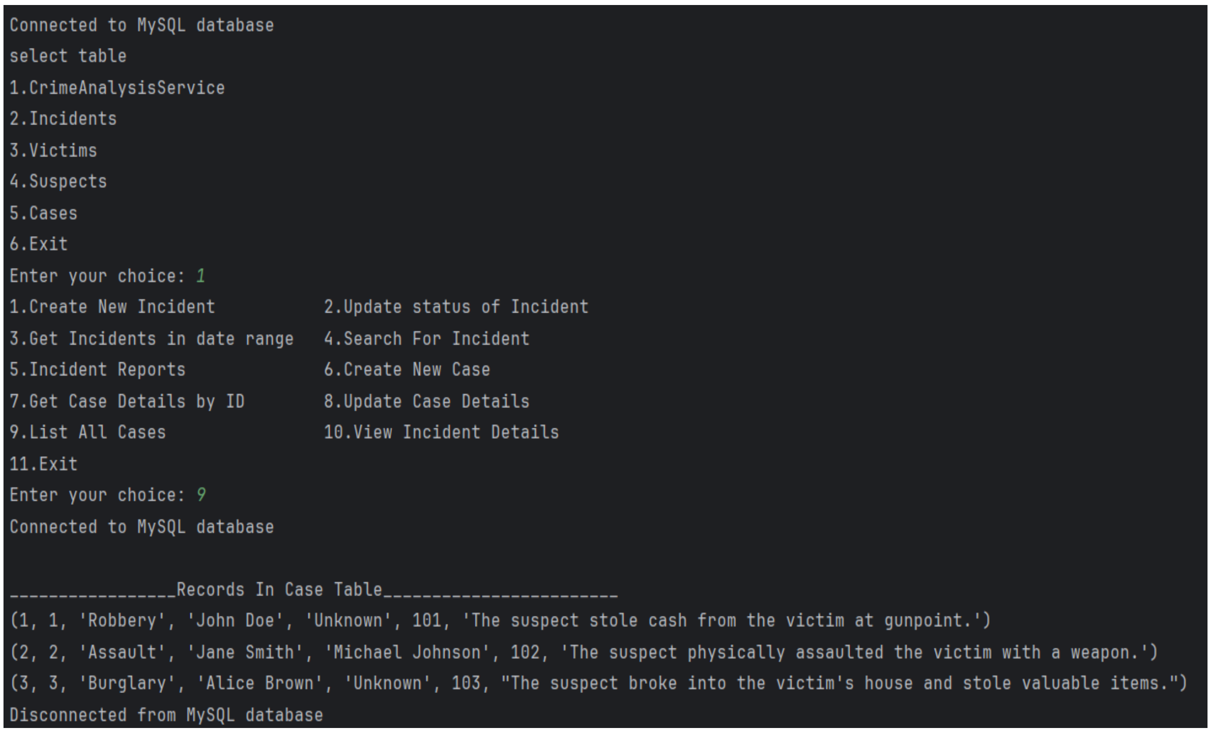
Create class named MainModule with main method in main package.

Trigger all the methods in service implementation class

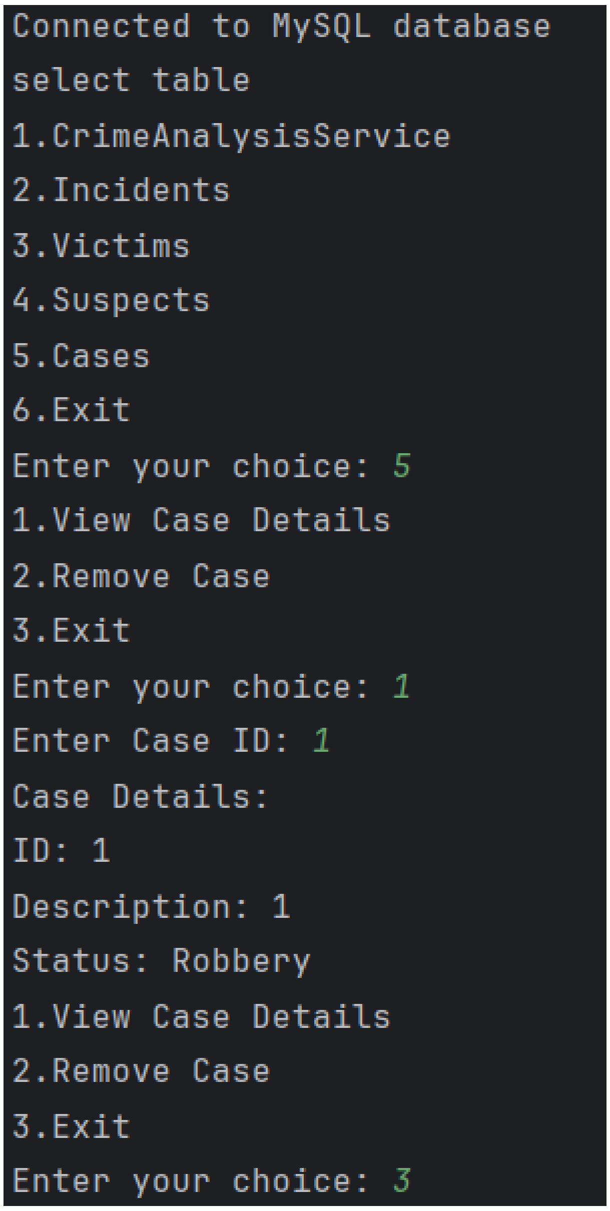
```
from dao.ICrimeAnalysisService import  ICrimeAnalysisService
from dao.Victims import  Victims
from dao.Suspect import Suspect
from dao.incidents import incidents
from dao.Icase import ICase
from MyExceptions.IncidentNumberNotFoundException import
IncidentNumberNotFoundException
condition = True
create = True
try:

    while condition:
        incident2 = ICrimeAnalysisService("localhost", "root", "root",
"3306", "crime_reporting")
        incident1=incidents()
        victim1=Victims()
        suspect1=Suspect()
        icase1 = ICase()
        print("select table")

print("1.CrimeAnalysisService\n2.Incidents\n3.Victims\n4.Suspects\n5.Cases\
n6.Exit")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            while True:
                print("1.Create New Incident          \t2.Update status of
Incident\n3.Get Incidents in date range\t"
                      "4.Search For Incident\n5.Incident Reports
\t\t6.Create New Case\n7.Get Case Details by ID\t"
                       "  \t8.Update Case Details\n9.List All Cases
\t\t\t10.View Incident Details\n11.Exit")
                choice = int(input("Enter your choice: "))
                if choice == 1:
                    incident2.create_incident()
                elif choice == 2:
                    incident2.update_incident_status()
                elif choice == 3:
                    start_date=input("Enter start date in yyyy-mm-dd: ")
                    end_date = input("Enter end date in yyyy-mm-dd: ")

incident2.get_incidents_in_date_range(start_date,end_date)
                elif choice == 4:
                    Incident_type=input("Enter Incident type: ")
                    incident2.search_incidents(Incident_type)
                elif choice == 5:
                    incidentid=input("Enter Tncident id: ")
                    incident2.generate_incident_report(incidentid)
                elif choice == 6:
                    incident2.create_case()
                elif choice == 7:
                    Caseid=int(input("enter Case id: "))
```

```python
                    incident2.CaseDetails(Caseid)
                elif choice == 8:
                    caseid = int(input("Enter Case id: "))
                    incident2.updateCaseDetails(caseid)
                elif choice == 9:
                    incident2.view_allcases()
                elif choice == 10:
                    incident2.select()
                else:
                    break
        elif choice == 2:
            while True:
                print("1.View Incident Details\n2.Remove Incident\n3.Exit")
                choice = int(input("Enter your choice: "))
                if choice == 1:
                    incident1.select()
                elif choice == 2:
                    incident1.delete()
                else:
                    break
        elif choice == 3:
            while True:
                print("1.Add Victim\n2.View Victims Details\n3.Remove
Victim\n4.Exit")
                choice = int(input("Enter your choice: "))
                if choice == 1:
                    victim1.add_victim()
                elif choice == 2:
                    victim1.select()
                elif choice == 3:
                    victim1.delete()
                else:
                    break
        elif choice == 4:
            while True:
                print("1.Add Suspect\n2.View Suspects Details\n3.Remove
Suspect\n4.Exit")
                choice = int(input("Enter your choice: "))
                if choice == 1:
                    suspect1.add_suspects()
                elif choice == 2:
                    suspect1.select1()
                elif choice == 3:
                    suspect1.delete()
                else:
                    break
        elif choice == 5:
            while True:
                print("1.View Case Details\n2.Remove Case\n3.Exit")
                choice = int(input("Enter your choice: "))
                if choice == 1:
                    icase1.select_case()
                elif choice == 2:
                    icase1.delete_case()
                else:
                    break
        else:
            break
except IncidentNumberNotFoundException as e:
    print(e)
```

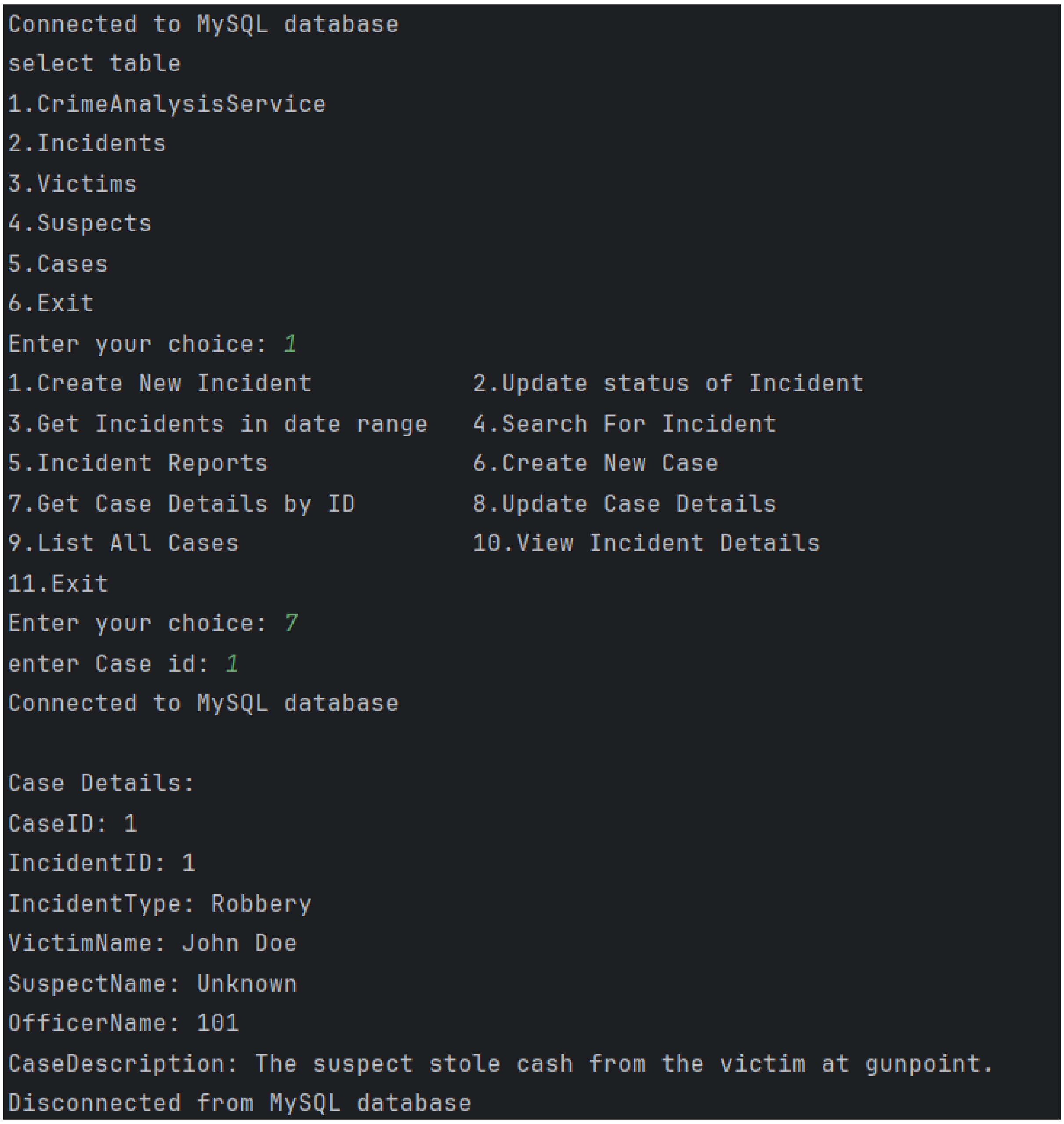
```python
except Exception as e:
    print(e)
```



```
Connected to MySQL database
select table
1.CrimeAnalysisService
2.Incidents
3.Victims
4.Suspects
5.Cases
6.Exit
Enter your choice: 1
1.Create New Incident          2.Update status of Incident
3.Get Incidents in date range   4.Search For Incident
5.Incident Reports             6.Create New Case
7.Get Case Details by ID       8.Update Case Details
9.List All Cases               10.View Incident Details
11.Exit
Enter your choice: 9
Connected to MySQL database

_________________Records In Case Table________________________
(1, 1, 'Robbery', 'John Doe', 'Unknown', 101, 'The suspect stole cash from the victim at gunpoint.')
(2, 2, 'Assault', 'Jane Smith', 'Michael Johnson', 102, 'The suspect physically assaulted the victim with a weapon.')
(3, 3, 'Burglary', 'Alice Brown', 'Unknown', 103, "The suspect broke into the victim's house and stole valuable items.")
Disconnected from MySQL database
```



```
Connected to MySQL database
select table
1.CrimeAnalysisService
2.Incidents
3.Victims
4.Suspects
5.Cases
6.Exit
Enter your choice: 5
1.View Case Details
2.Remove Case
3.Exit
Enter your choice: 1
Enter Case ID: 1
Case Details:
ID: 1
Description: 1
Status: Robbery
1.View Case Details
2.Remove Case
3.Exit
Enter your choice: 3
```



```
Connected to MySQL database
select table
1.CrimeAnalysisService
2.Incidents
3.Victims
4.Suspects
5.Cases
6.Exit
Enter your choice: 1
1.Create New Incident           2.Update status of Incident
3.Get Incidents in date range   4.Search For Incident
5.Incident Reports              6.Create New Case
7.Get Case Details by ID        8.Update Case Details
9.List All Cases                10.View Incident Details
11.Exit
Enter your choice: 7
enter Case id: 1
Connected to MySQL database

Case Details:
CaseID: 1
IncidentID: 1
IncidentType: Robbery
VictimName: John Doe
SuspectName: Unknown
OfficerName: 101
CaseDescription: The suspect stole cash from the victim at gunpoint.
Disconnected from MySQL database
```

10. Unit Testing

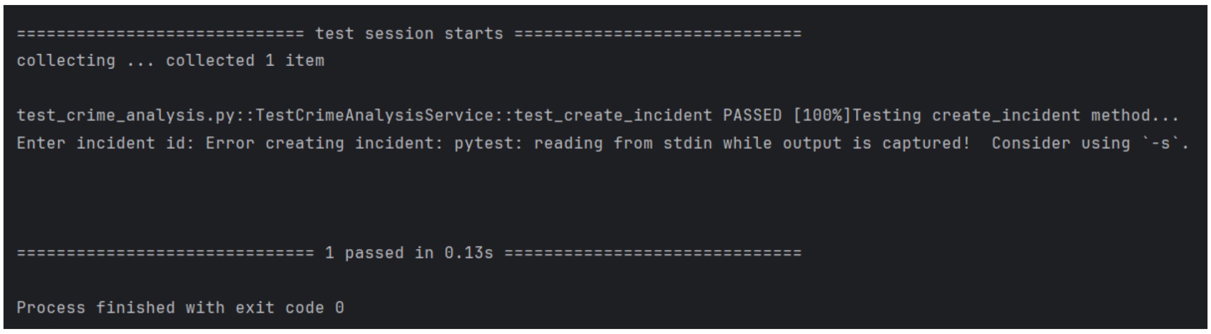
Creating JUnit test cases for a Crime Analysis and Reporting System is essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of JUnit test cases for various components of the system:

1. Incident Creation:

• Does the createIncident method correctly create an incident with the provided attributes?

• Are the attributes of the created incident accurate?

```python
import pytest
from dao.ICrimeAnalysisService import ICrimeAnalysisService

@pytest.fixture
def service():
    # Initialize the service
    return ICrimeAnalysisService()

@pytest.fixture
def incident_data():
    # Sample incident data
    return {
        'incident_id': 4,
        'incident_type': 'Robbery',
        'incident_date': '2023-01-10',
        'location': '123 Main St, City, Usa',
        'description': 'Armed robbery at a convenience store.',
        'status': 'Open',
        'victim_id': 4,
        'suspect_id': 4
    }

def test_create_incident(service, incident_data):
    # Test createIncident method
    created = service.create_incident(**incident_data)
    assert created

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



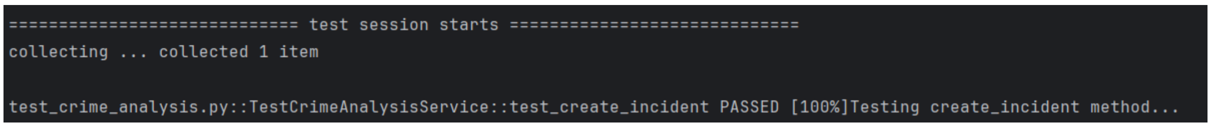
```
============================= test session starts ==============================
collecting ... collected 1 item

test_crime_analysis.py::TestCrimeAnalysisService::test_create_incident PASSED [100%]Testing create_incident method...
Enter incident id: Error creating incident: pytest: reading from stdin while output is captured!  Consider using `-s`.



============================== 1 passed in 0.13s ==============================

Process finished with exit code 0
```

2. Incident Status Update:

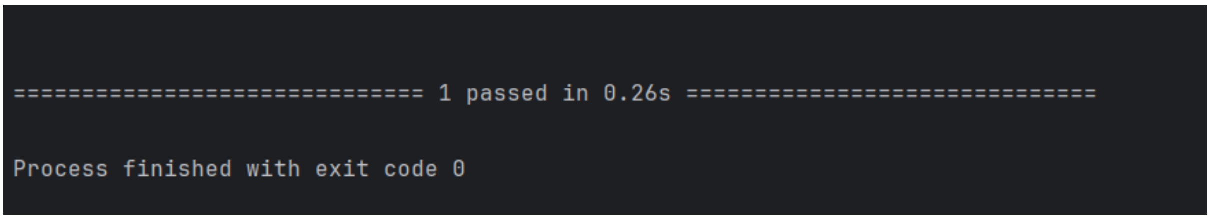
• Does the updateIncidentStatus method effectively update the status of an incident?

• Does it handle invalid status updates appropriately?

```python
import unittest
from dao import ICrimeAnalysisService
from entity import Incidents
class TestCrimeAnalysisService(unittest.TestCase):
def test_update_incident_status(self):
        # Create a test incident data
        incident_data = {
            'incident_type': 'Robbery',
            'incident_date': '2023-01-10',
            'location': '123 Main St, City, Usa',
            'description': 'Armed robbery at a convenience store.',
            'status': 'Open',
            'victim_id': 3,
            'suspect_id': 3
        }

        # Create an Incidents object without an incident_id attribute
        incident = Incidents.Incidents(**incident_data)

        # Call create_incident method to insert the incident into the
database
        created = self.service.create_incident(incident)

        # Check if incident creation was successful
        self.assertTrue(created)

        # Get the incident ID from the created incident using some method,
assuming it's generated during insertion
        # For demonstration purposes, let's assume get_incident_id()
returns the generated incident ID
        incident_id = incident.get_incident_id()

        # Test updateIncidentStatus method
        new_status = 'Closed'
        updated = self.service.update_incident_status(incident_id,
new_status)

        # Retrieve the updated incident
        updated_incident = self.service.get_incident_details(incident_id)
        print(updated_incident)

        # Verify that the status has been updated correctly
        assert updated_incident[5] == new_status

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



```
============================= test session starts ==============================
collecting ... collected 1 item

test_crime_analysis.py::TestCrimeAnalysisService::test_create_incident PASSED [100%]Testing create_incident method...
```



```


============================== 1 passed in 0.26s ===============================


Process finished with exit code 0
```