Object-Oriented Programming Project : Patient Data Analysis and Prediction System

Presented by:

- Ranim Sayahi
- Ghaith Dkhili
- Safa Zaghdoudi

Senior Level :
BA/IT

Supervised by:
Professor Amani Azouz

Academic year :
2024/2025

Table of Content

General Introduction:

Our project focuses on the detection and analysis of diabetes using data-driven methods. The system utilizes a dataset comprising existing patients' health records, including key medical indicators, such as glucose levels, blood pressure, insulin, BMI, and their diabetic status (whether they are diabetic or not).

The project aims to analyze this data to train a machine learning model capable of identifying patterns and relationships between the various health parameters and diabetes. After training, the model is tested for accuracy and performance to ensure reliable predictions. Once validated, the model can then be applied to new patient data, where individuals input their personal health information to determine their likelihood of being diabetic. The system provides comparisons with existing data to enhance interpretability and trustworthiness.

This predictive model offers a powerful tool for doctors and the healthcare system, facilitating early detection of diabetes, supporting decision-making, and improving patient outcomes. By automating the analysis process, the system helps streamline the workflow, saves time, and enhances the efficiency of diabetes screening, making it an effective solution in modern healthcare.

1. Project Requirements:

Objective:

To develop a data analysis and management system focused on diabetes data using an object-oriented programming (OOP) approach. The system ensures patient data privacy and provides tools for data analysis, visualization, and machine learning-based prediction.

Functional Requirements:

- **Data Ingestion**:
  - Import data from CSV files for both personal and medical records.
  - Parse and store the data securely.
- **Data Privacy**:
  - Keep patient identity confidential by separating personal data (names, surnames) from medical data using unique patient IDs.
- **Data Analysis**:
  - Perform basic statistical analyses (e.g., average glucose levels, BMI distributions).
- **Machine Learning Integration**:
  - Implement a decision tree to predict diabetes risk based on patient medical records.
- **Data Visualization:**
  - Visualize our patient's data for in depth data analysis and understanding the overall composition of our data via different visuals.

Non-Functional Requirements:

- **User-Friendliness**:
  - Provide a modular and maintainable system using OOP principles.
- **Scalability**:
  - The system should handle larger datasets in the future.
- **Security**:
  - Protect patient data through separation and encapsulation of sensitive information.

2. Technologies used:

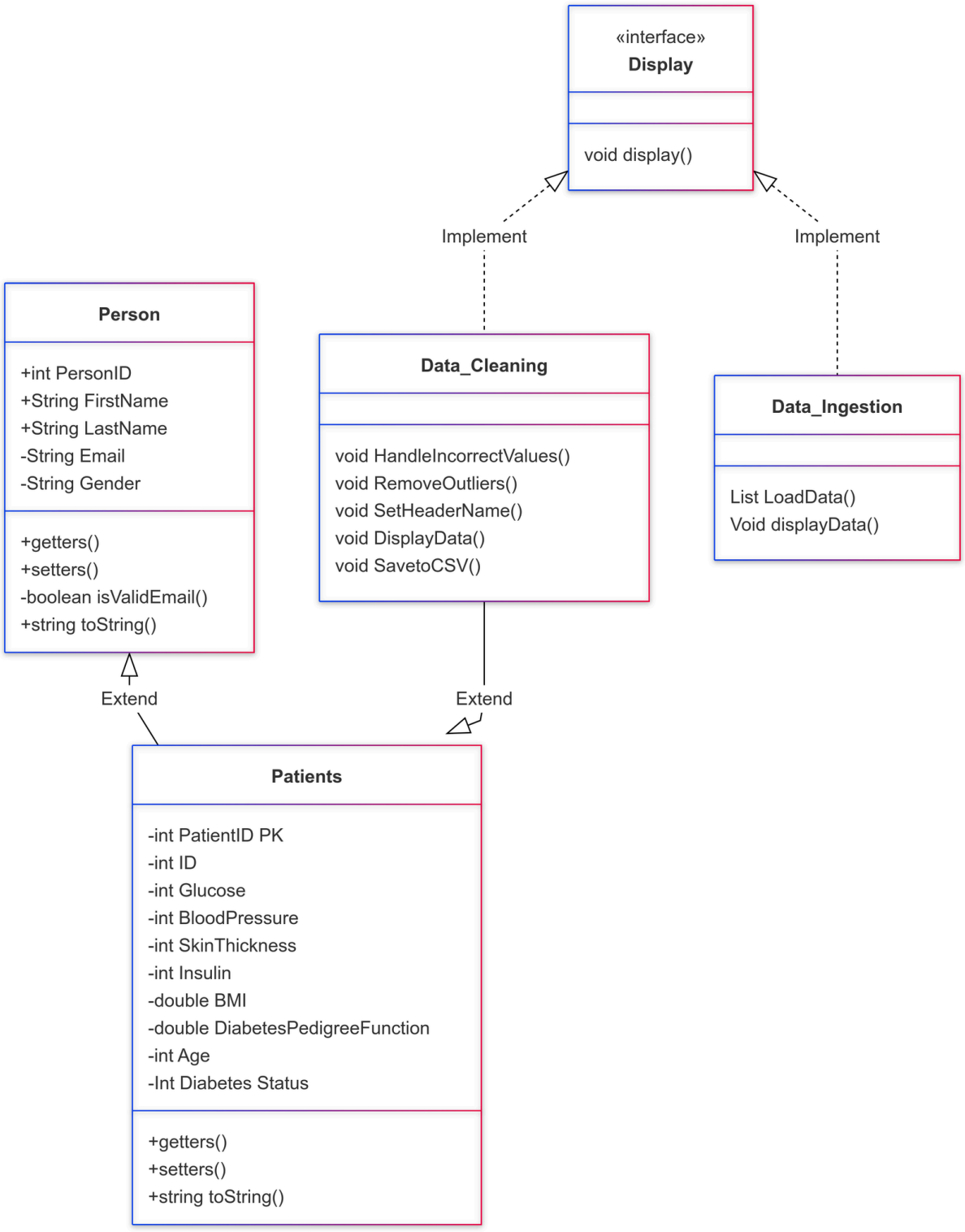
- **Database Layer**:

PostegreSQL was used to store and handle the database consisting of 2 tables: Person and Patient

- **Backend Layer**:

Netbeans was used to define and implement all needed classes and interfaces while respecting all the OOP concepts
Some libraries used are: java.awt, org.jfree, javax.swing, java.util…….;

3. Classes and Interfaces:



- **Class Person:**

  -Description: Stores personal information such as name, surname, and a unique patient ID.

  -Attributes:

  - `int ID`
  - `string FirstName`
  - `string LastName`
  - `string Email`
  - `string Gender`

  -Methods:

  - Getters and setters for attributes.
  - toString
  - isValidEmail

- **Class Patients:**

  -Description: Stores medical data related to patients and inherits the attributes of the class person

  -Attributes:

  - `int PatientID`
  - `int Glucose`
  - `int BloodPressure`
  - `int SkinThickness`
  - `int Insulin`
  - `int Age`
  - `int Outcome`
  - `double BMI`
  - `double DiabetesPrediggreeFunction`

  -Methods:

  - Getters and setters for attributes.
  - toString

- **Interface Display:**

-Description: An interface that contains an abstract method for security of implementation and to allow multiple inheritance.

-Methods:

- Void Display()

- **Class Data_Ingestion:**

-Description:A class that serves the purpose of fetching the initial datasets into

-Methods:

- List LoadData()
- Void DisplayData()

- **Class Data_Cleaning:**

-Description:A class that serves the purpose of cleaning the data imported into netbeans and then exporting it into a csv file.

-Methods:

- Void HandleIncorrectValues()
- RemoveOutliers()
- SetHeaderName()
- DisplayData()

- **Class DBConnect:**

-Description:This class serves the purpose of making the connection between PostgreSQL and Netbeans

-Attributes:

- `static final String URL`
- `static final String USER`
- `static final String PASSWORD`

-Methods:

- connect()
- getAllPersonsAsObjects()
- getAllPatientsAsObjects()

- **Class Data_Analysis:**

-Description:A class that serves the purpose of analysing the data and gaining insights

-Methods:

-classifyBMI(Patients patient)

-calculateRiskScore(Patients patient)

-classifyRisk(Patients patient)

- calculateHealthScore(Patients patient)

-classifyAge(Patients patient)

-classifyBloodPressure(Patients patient)

-classifySkinThickness(Patients patient)

-classifyInsulinLevel(Patients patient)

-classifyGlucoseLevel(Patients patient)

- mean(List<Double> data)

-median(List<Double> data)

-frequencyDistribution(List<String> data)

- **Class Data_Visualization:**

-Description:A class that serves the purpose of Visualizing the data for in-depth analysis of the data and understanding of its composition.

-Methods:

-createAgeGroupDistributionChart()

-createSicknessDistributionChart()

-ComparePatients(int patientId1, int patientId2)

- displayPatientComparisonTable(int patientId1, int patientId2)

-createBMIDistributionCharts()

- **Class Machine_learning:**

-Description:The machine_learning class splits our dataset into 2 parts (80% for training where a decision tree classifier model is used and 20% is for testing). Then using the testing part the accuracy of our model is calculated. The model is then used to predict a new patient's status based on their entered data.

-Methods:

-trainAndPredictDiabetesRisk()

-convertPatientsListToDataFrame()

-prepareDataForTraining()

-VectorAssembler.setInputCols()

-VectorAssembler.setOutputCol()

-Dataset<Row>.randomSplit()

-DecisionTreeClassifier.fit()

-model.transform()

-MulticlassClassificationEvaluator.evaluate()

-functions.col()

-Dataset<Row>.withColumn()

-Dataset<Row>.groupBy()

- **Class Main:**

-Description:This class serves the purpose of testing individual components and methods of the program by creating instances of classes.

4. Relationships between classes:

- **Person <|-- Patients : Extend:**

This inheritance relationship, “Person <|-- Patients : Extend” means that Patients is a subclass of Person, and it inherits all the attributes and behaviors (fields and methods) of the Person class.

1. Person (Superclass):
    - Represents general attributes and methods common to all people.
2. Patients (Subclass):
    - Represents a more specific type of person: someone who is a patient. It inherits the common attributes and methods from Person (like name, age, etc.).

=>This Inheritance allows code reuse by avoiding duplication. Instead of defining common attributes in both classes, you define them once in the Person class, and Patients can reuse them.

- **Display <|.. Data_Cleaning : Implement**

The relationship “Display <|.. Data_Cleaning : Implement” indicates that the Data_Cleaning class implements the Display interface. In Java, an interface defines a set of methods that a class must fulfill when it implements the interface.

1. Display (Interface):
    - It defines an abstract method. And it represents a contract that any implementing class must adhere to.
2. Data_Cleaning (Class):
    - It provides the actual implementation of the method defined in the Display interface.

=>Interfaces enforce a contract between the interface and the implementing class.They allow multiple classes to implement the same set of methods differently, enabling polymorphism.

- **Display <|.. Data_Ingestion : Implement:**

The relationship “Display <|.. Data_Ingestion : Implement” indicates that the Data_Ingestion class implements the Display interface. It has the same spirit as the Data_Cleaning class and interface Display’s relationship.

- **Data_Cleaning --|> Patients : Extend:**

The relationship “Data_Cleaning --|> Patients : Extend” means that the Data_Cleaning class inherits from the Patients class.

This signifies that Data_Cleaning is a specialized version of Patients, and it gains access to all the fields (attributes) and methods of the Patients class. Additionally, the Data_Cleaning class can:

-Add its own specific attributes and methods.

-Override methods from the Patients class if needed.:

1. Patients(Base Class)
    - Represents a general blueprint for patients, likely containing: Attributes such as name, age, BMI, bloodPressure, etc. and Methods like calculateRiskScore() or classifyBMI().
2. Data_Cleaning(subclass)
    - Extends the Patients class, meaning: It inherits all patient-related attributes and methods. But It adds its own functionality related to cleaning or preprocessing patient data.

5. OOP Concepts used in the project:

Throughout our project’s implementation we tried to incorporate different OOP main concepts to provide a structured and efficient way to organize and manage code.

1. **Inheritance:**

Definition: Inheritance allows a class to **inherit** attributes and behaviors from another class, promoting **hierarchical classification** and reducing duplication.
Implementation in our code: We used two Inheritances:

- *“Patients” inherits from “person”:* Representation of a more specific type of person: someone who is a patient inheriting the common attributes and methods from Person (like name, gender, etc.) and adding its own methods and attributes to it.
- *“Data_Cleaning” inherits from “Patients”:* Extension of the Patients’ class by inheriting all patient-related attributes and methods and adding functionalities related to cleaning and preprocessing the patients’ data.

2. **Encapsulation:**

Definition: Encapsulation hides the internal details of objects and only exposes what is necessary using **getter** and **setter** methods. This ensures that an object's state is protected from unintended interference or misuse.
Implementation in our code: Encapsulation is a fundamental concept of Object-Oriented Programming that we effectively implemented in our project by defining all attributes as private and providing controlled access through getter and setter methods. This approach

ensures that the internal state of each object is protected from unauthorized or unintended modifications, promoting data security and integrity.

3. **Abstraction:**

Definition:OOP allows you to focus on **what an object does** rather than **how it does it** by using **abstract classes** and **interfaces**. This simplifies the design and makes it easier for other developers to use the code.
Implementation in our code: We achieved abstraction through the use of the “Display” interface. The interface defines the contract or blueprint for the methods that need to be implemented, without specifying how these methods should work. This allows us to focus on "what" the classes should do, rather than "how" they do it.

4. **Polymorphism:**

Definition:OOP supports **polymorphism**, enabling one interface to be used for multiple types of objects.This allows for **dynamic method binding**, where the appropriate method is determined at runtime.
Implementation in our code: We demonstrated polymorphism through the implementation of the “*Display*”interface by two distinct classes, “*Data_Cleaning*” and “*Data_Ingestion*”. The Display interface defines a set of methods that both implementing classes are required to override, but each class provides its own specific implementation of these methods based on its unique functionality.

General Conclusion

In conclusion, our project, which focuses on the detection and analysis of diabetes using data-driven methods, demonstrates the effective application of key Object-Oriented Programming (OOP) principles—encapsulation, inheritance, polymorphism, and abstraction—in building a modular, flexible, and maintainable system. By utilizing a dataset of patients' health records, including crucial medical indicators like glucose levels, blood pressure, insulin, BMI, and diabetic status, our system aims to provide valuable insights into the risk and management of diabetes.

Encapsulation ensured secure handling of sensitive health data by defining attributes as private and allowing access through controlled getter/setter methods. Inheritance facilitated efficient code reuse by allowing the creation of specialized classes that extend the functionality of base classes. Polymorphism enhanced the system's flexibility by allowing the use of multiple implementations of shared interfaces, adapting to various data-processing needs. Abstraction simplified complex processes, focusing on high-level functionalities and hiding the implementation details.

Through the combination of these OOP principles, our project provides a scalable and reliable system for diabetes detection and analysis. The modular design ensures that the system can be easily updated or extended as new data or features become available. This approach highlights the importance of OOP in building data-driven software solutions, ensuring a maintainable and extensible system for real-world healthcare applications.