Klasse io_data

Die Klasse „io_data“ dient der strukturierten Konvertierung, Ein- und Ausgabe der Patientendaten. Im Konstruktor werden Membervariablen definiert, die patientenbezogene Informationen, wie ID, Name, Geburtsdatum, Geschlecht, Kontaktdaten und medizinische Details umfassen.

Die Implementierung der Klasse enthält eine Methode zur Altersberechnung anhand des Geburtsdatums und Funktionen zur bidirektionalen Konvertierung des Datums zwischen den Formaten QDate und QString.

Des Weiteren verwaltet Sie die Datenflüsse für der Datenein- und -ausgabe. Dies erfolgt mithilfe der Klasse „database“, die die Datenbankanbindung realisiert und die Patientendaten in der Datenbank abspeichert, bzw. liest. Zudem sind die Funktionalitäten zum Einlesen von Patientendaten aus CSV-Dateien und Exportierung der gespeicherten Patientendaten in einer CSV-Datei enthalten.

Diese Klasse **„io_data“** ist zentraler Bestandteil des Projektes und enthält folgende Methoden:

- **int returnAge()**
  Berechnet das Alter basierend auf dem Geburtsdatum des Patienten

- **static QDate convertQStringToQDate(const QString datumString)**
  Konvertiert ein Datum vom QString-Format in ein QDate-Objekt

- **static QString convertQDateToQString(const QDate datum)**
  Wandelt ein QDate-Objekt in eine QString um.

- **static void CSVeinlesen(QString pfad, Database &database)**
  Liest Patientendaten aus einer CSV-Datei ein und speichert sie in die Datenbank

- **static void CSVerstellen(QString pfad, Database &database)**
  Erstellt eine CSV-Datei aus den in der Datenbank gespeicherten Patientendaten

int io_data::returnAge()

Die Methode returnAge() berechnet das Alter eines Patienten basierend auf dem Geburtsdatum und dem aktuellen Datum und gibt dieses als int zurück.

Sie wandelt den QString der Membervariable „geburt“ über einen Standardstring in einen stream um und zerlegt diesen an den enthaltenen Punkten in drei Variablen „int tag“, „int monat“ und „int jahr“.

Anschließend wird die Systemzeit geladen und durch Subtraktion des Jahres das Alter berechnet. Zudem wird über eine if-Abfrage das Alter korrigiert, falls das aktuelle Datum vor dem Geburtstag liegt um das korrekte tagesaktuelle Alter zurückzugeben.

QDate io_data::convertQStringToQDate(const QString datumString)

convertQStringToQDate() prüft den übergebenen String auf Gültigkeit (Format „dd.MM.yyyy“) und gibt bei Erfolg das Datum als QDate zurück. Ansonsten wird ein ungültiges QDate zusammen mit einer qWarning zurückgegeben.

QString io_data::convertQDateToQString(const QDate datum)

convertQDateToQString() ist die inverse Funktion zu convertQStringToQDate() und wandelt das QDate in einen QString im Format „dd.MM.yyyy“ um. Dieses wird zur z. B. zur Speicherung in der Datenbank nach dem Bearbeiten eines Datensatzes benötigt.

void io_data::CSVeinlesen(QString pfad,Database &database)

Die Methode CSVeinlesen() ist mit Try-Catch-Blöcken realisiert, um die Programmstabilität sicherzustellen. Sie öffnet die ausgewählte CSV-Datei, liest dessen Patientendaten zeilenweise ein und validiert diese mittels Regex. Anschließend werden die validierten Daten in der Datenbank abgespeichert. Dabei wird die erste Zeile, die die Spaltenüberschrift enthält übersprungen.

Damit nur vollständige und korrekte Datensätze in der Datenbank abgelegt werden, wird nach dem erfolgreichen Einlesen jeder Zeile der Patientendatensatz mit regex validiert, und die Anzahl an Werten überprüft, bevor er über die Klasse „database“ mit der Methode „insertPatient“ der Datenbank hinzugefügt wird. Schlägt hierbei regex fehl oder stimmt die Anzahl der eingelesenen Werte nicht, wird eine Catch ausgelöst und ein qDebug ausgegeben.

Nach dem vollständigen Einlesen der Datei, wird diese wieder geschlossen, um Speicherzugriffsproblemen, Programmstabilitätseinbrüche und Bugs vorzubeugen.

void io_data::CSVerstellen(QString pfad, Database &database)

Die Methode CSVerstellen() exportiert die Patientendaten aus der Datenbank in eine CSV-Datei. Sie erstellt (bzw. überschreibt) die CSV-Datei und fügt in dessen erste Zeile die Spaltenüberschrift der Patientendaten ein. Anschließend werden die Patientendatensätze der Datenbank nacheinander durchgegangen und in die CSV-Datei geschrieben, bevor die Datei wieder geschlossen wird.

Um die Stabilität des Programms zu gewährleisten, ist diese Methode ebenfalls in einen Try-Catch-Block eingebettet. Dies stellt sicher, dass mögliche Fehler durch Dateioperationen auf Systemebene, die durch Systeminteraktionen oder das Verhalten anderer im Hintergrund laufender Anwendungen entstehen können, abgefangen und behandelt werden.

Widgetverbindung, Programmierung und Anpassungen

Im mainwindow.cpp werden durch connect-Befehle die Signale mit den Slots (auszuführenden Methoden) verbunden, um beispielsweise beim Auswählen eines Patienten in der Tabelle die gesammelten Daten des Patientendatensatzes im Ausgabefenster (TextEdit - Hauptfenster rechts unten) zu aktualisieren. Die Slot Methode wird auch nach dem Bearbeiten eines Datensatzes aufgerufen, um die Ausgabe im Hauptfenster zu aktualisieren.

Die „on_..._clicked()“ Methoden werden ausgeführt, wenn die verbundenen Schaltflächen betätigt werden. So wird zum Beispiel beim Speichern Knopf (Disketten Symbol) über den QFileDialog der Dateipfad für die speichern Funktionalität eingelesen. Beim Aktivieren der Öffnen Schaltfläche wird der ausgewählte Dateipfad an die Methode „CSVerstellen“ der Klasse „io_data“ übergeben.

Ist das „Datensatz anzeigen“ Fenster geöffnet und wird dort der Button „Datensatz bearbeiten“ ausgewählt, sorgt die dadurch aufgerufene Methode dafür, dass nach dem Öffnen des “Datensatz bearbeiten“ Fensters der Destruktor des Datensatz anzeigen Objektes aufgerufen und somit das vorherige Fenster geschlossen wird.

Eine besondere Herausforderung war hierbei die Verriegelung zwischen dem „Datensatz anzeigen“ Fenster und dem „Datensatz bearbeiten“ Fenster, das ebenfalls über „Datensatz hinzufügen“ geöffnet werden kann. Die zuverlässige Verriegelung und damit das Verhindern, des doppelten Öffnens um zu vermeiden zur selben Zeit mehrmals denselben Datensatz zu bearbeiten wurde Mithilfe von Pointer

und PointerPointer gelöst. „Datensatz anzeigen“ und “Datensatz bearbeiten“ verhindert zuverlässig und klassenübergreifenden mithilfe von If-Abfragen der Pointer, dass ein Fenster des jeweiligen Types doppelt geöffnet werden kann. Hierfür wird der Destruktor aktiv mit einbezogen und aufgerufen, wenn Fenster geschlossen werden, damit werden auch beim Schließen des Hauptfensters die Objekte zerstört, Fenster geschlossen und Pointer zu Nullpointer.

Für die „Nutzer anlegen“ Klasse wurde der Pointer als PointerPointer übergeben, da damit das mainwindow nicht zusätzlich inkludiert werden muss. Somit läuft die Anwendung stabil, handhabt und unterdrückt Eingabefehler.

Zudem wurden sämtliche Widgets auf „Grid Layout“ umgestellt, damit auch beim Großziehen der Fensterinhalt flexibel mitskaliert wird. Die Scrollbalken werden dadurch ebenfalls skaliert und können bei ausreichender Größe verschwinden, wie beispielsweise beim „Datensatz bearbeiten“ Fenster.

Am Ende wurde die Anwendung noch mit QMessageBoxen abgerundet, die Warnungen bei unzulässigen Operationen (z. B. dem Versuch eines doppelten Aufrufs für Datensatz bearbeiten, anzeigen oder Nutzer anlegen) des Benutzers ausgibt.

„if (mw->Datensatz_bearbeiten_fenster != nullptr) { … }“

QMessageBox::warning(this, "Warnung", "Bereits ein Fenster datensatz_anzeigen_fenster offen!");

Pointer für Fensterverrigelung, damit nicht mehrere Bearbeiten oder Anzeigefenster gleichzeitig angezeigt werden können.

```
Datensatz_bearbeiten* Datensatz_bearbeiten_fenster = nullptr;
datensatz_anzeigen* datensatz_anzeigen_fenster = nullptr;
```

```
void datensatz_anzeigen::on_pushButton_4_clicked()
{
  //Berechtigung wird überprüft
  if(mw->akt_user!=nullptr && mw->akt_user->permission==3){
    QMessageBox::warning(this,"Fehler","Sie haben nur eine Leseberechtigung");
    return;
  }
```