h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Programmieren II

Praktikum 4: Smart Pointer, Container, Funktoren

Wintersemester 2023

Prof. Dr. Arnim Malcherek

Allgemeine Hinweise zum Praktikum:

- Bereiten Sie die Aufgaben unbedingt zu Hause oder in einem freien Labor vor. Das beinhaltet:
  - Entwurf der Lösung
  - Codieren der Lösung in einem Qt-Creator-Projekt
- Die Zeit während des Praktikums dient dazu, die Lösung testieren zu lassen sowie eventuelle Korrekturen vorzunehmen.
- Das Praktikum dient auch zur Vorbereitung der praktischen Klausur am Ende des Semesters. Versuchen Sie also in Ihrem eigenen Interesse, die Aufgaben selbständig nur mit Verwendung Ihrer Unterlagen bzw. Ihres bevorzugten C++-Buches und ohne Codefragmente aus dem Netz zu lösen.
- Die Lösung wird nur dann testiert, wenn
  - sie erklärt werden kann bzw. Fragen zur Lösung beantwortet werden können.
  - das Programm ablauffähig und die Lösung nachvollziehbar ist.
  - die Hinweise oder Einschränkungen aus der Aufgabenstellung befolgt wurden.
- Zur Erinnerung hier noch einmal die Regeln des Praktikums, die schon in der Vorlesung besprochen wurden:
  - Sie können in Gruppen arbeiten. Ich empfehle 2er Gruppen, damit Sie genügend selbst programmieren. Die Abgabe erfolgt aber einzeln. Sie müssen das Coding also im Detail kennen und verstehen.
  - Ein Testat gibt es nur zum jeweiligen Termin.
  - Es gibt keine Noten. Die Bewertung ist lediglich erfolgreich / nicht erfolgreich.
  - Das Praktikum ist Zulassungsvoraussetzung für die Klausur. Hierfür müssen alle fünf Praktikumsübungen testiert sein.

Lernziele:

- Smart Pointer
- Container
- Funktionsobjekte

Aufgabe 1

(a) In der Datei *bookingsPraktikum4.json* sind GPS-Koordinaten für alle Orte (Flughäfen, Mietwagen-Stationen, Hotels, Bahnhöfe) dazugekommen. Lesen Sie diese mit ein. Sie werden später in Aufgabe 3 verwendet. Natürlich benötigen Sie dafür zusätzliche Attribute in Ihren Klassen. Die GPS-Koordinaten müssen aber nicht in Ihren Detailfenstern angezeigt werden, und sie müssen auch nicht änderbar sein (genaueres dazu in Aufgabe 3).

(b) Fügen Sie eine Funktion im UI ein, mit der Sie das Einlesen der Datei *iatacodes.json* anstoßen können, oder lesen Sie die Datei direkt nach der Datei mit den `Bookings` ein. Die Datei *iatacodes.json* enthält Details zu den Flughafencodes (siehe Abbildung 1).


```
{
  "name": "Montreal / Pierre Elliott Trudeau International Airport",
  "iso_country": "CA",
  "municipality": "Montréal",
  "iata_code": "YUL"
},
{
  "name": "Vancouver International Airport",
  "iso_country": "CA",
  "municipality": "Vancouver",
  "iata_code": "YVR"
},
```


Abbildung 1: Auszug aus JSON-Datei zu Flughafencodes

Legen Sie eine Klasse oder Struktur mit Namen `Airport` an, in der Sie die Daten für einen Flughafen speichern können. Fügen Sie jetzt der Klasse `TravelAgency` einen Container vom Typ `std::map<iata_code, shared_ptr<Airport»` oder `QMap<iata_code,shared_ptr<Airport»` hinzu, und speichern Sie den Inhalt der Datei in diesem Container ab.

(c) Ändern Sie die Detailanzeige in Ihrem User Interface so ab, dass zusätzlich zum Flughafencode auch der Name des Flughafens angezeigt wird. Nur der Flughafencode soll änderbar sein. Wenn der/die User*in den Code ändert, soll geprüft werden, ob der eingegebene Code existiert und dann natürlich auch der richtige Name dazu angezeigt werden. Wenn der Code nicht existiert, geben Sie im Feld für den Namen des Flughafens `Ungültiger Iata-Code` in roter Farbe aus.

Aufgabe 2

Ersetzen Sie alle Zeiger auf `Booking`, `TravelAgency`, `Travel` und `Customer` durch Smart Pointer. Überlegen Sie selbst, ob Sie `unique_ptr` oder `shared_ptr` einsetzen. **Zeiger auf Klassen, die von `QObject` erben, müssen Sie nicht ersetzen** (falls Sie es doch tun

wollen, verwenden Sie `QPointer` oder `QSharedPointer`).

Aufgabe 3

(a) Alle Buchungen einer Reise sollen zusätzlich zur Anzeige in einem `TableWidget` auch mit ihren GPS-Koordinaten und einem Text auf einer Karte angezeigt werden. Bei einem Doppelklick auf eine Reise soll also nicht nur die Liste der Buchungen angezeigt werden, sondern gleichzeitig auch die Karte angezeigt werden.

- Für Flugbuchungen soll auf der Karte eine Linie vom Start- zum Zielflughafen angezeigt werden. Außerdem sollen die Namen der beiden Flughäfen angezeigt werden.
- Für Mietwagenbuchungen soll eine Linie angezeigt werden, wenn sich Abhol- und Rückgabestation unterscheiden. Sonst soll nur ein Punkt angezeigt werden. Zusätzlich soll der Name der Station und der Firma angezeigt werden.
- Für Hotelbuchungen soll ein Punkt und der Name des Hotels angezeigt werden (siehe Beispiel in Abbildung 2).
- Für Zugbuchungen soll eine Linie vom Startbahnhof über alle Verbindungsbahnhöfe bis zum Zielbahnhof angezeigt werden. Zu allen Bahnhöfen soll zusätzlich der Name angezeigt werden.

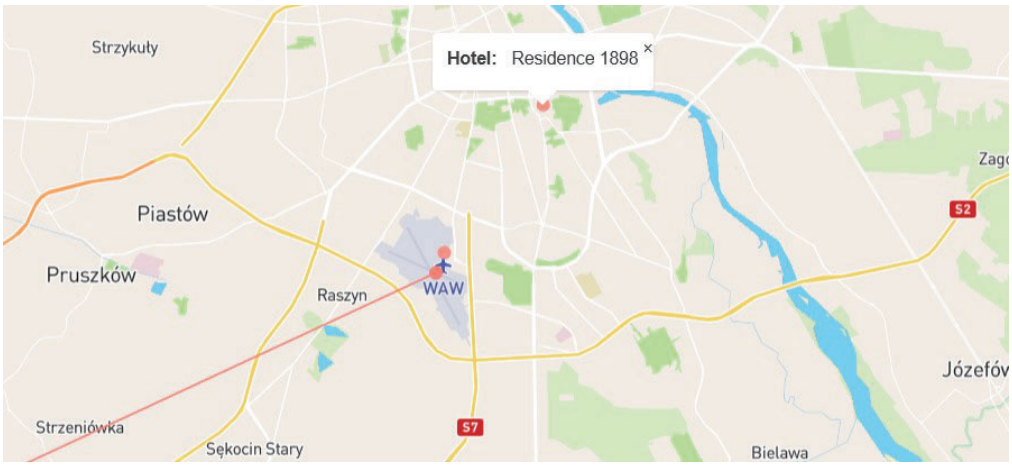

Abbildung 2: Beispiel zur Kartendarstellung

(b) Zur Darstellung auf einer Karte gibt es (mindestens) zwei Möglichkeiten:

- Sie rufen die Web-Site von Jennings Anderson mit validem GeoJson-Code auf. Eine URL können Sie z.B. über `QDesktopServices::openUrl` oder ein `QLabel` oder `QTextEdit` aufrufen.
  Beispiel: `http://townsendjennings.com/geo/?geojson={"type":"Point","coordinates":[3.1977,46.6841]}`
  (Für die Definition von GeoJson siehe `https://geojson.org` oder `https://de.wikipedia.org/wiki/GeoJSON`).
- Sie verwenden die QML-Komponente `MapView` und betten die Komponente über ein `QQuickWidget` in Ihr UI ein.

Die erste der beiden Möglichkeiten ist deutlich einfacher. Die zweite ist für diejenigen von Ihnen gedacht, die die Möglichkeiten von Qt etwas intensiver ausprobieren möchten.

Aufgabe 4

Fügen Sie Ihrem Projekt eine Klasse `Check` hinzu. Diese Klasse soll dazu dienen, die Daten aller Buchungen auf Konsistenz zu prüfen. Die Klasse wird im Konstruktor von `TravelAgencyUI` erzeugt. Damit `Check` auf alle Daten zugreifen kann, benötigt die Klasse einen Zeiger auf `TravelAgency` (`std::shared_ptr<TravelAgency>`).

- Die einzelnen Prüfungen werden über private Methoden der Klasse abgebildet. In diesem Praktikum müssen Sie nur eine Prüfmethode implementieren: Die Methode `bool checkTravelsDisjunct(QString& message)` soll prüfen, ob ein Kunde Reisen hat, die sich zeitlich überlappen.
  Beispiel:
  Kunde: Theodor Fontane
  Reise 1:
  Buchung 1: 01.03.2021 - 01.03.2021
  Buchung 2: 01.03.2021 - 05.03.2021
  Buchung 3: 05.03.2021 - 06.03.2021
  Reise 2:
  Buchung 1: 26.02.2021 - 27.02.2021
  Buchung 2: 27.02.2021 - 02.03.2021

  Falls mehrere Reisen eines Kunden überlappen wie in diesem Beispiel, soll die Methode `false` zurückgegeben, sonst `true`. Bei Ergebnis `false` soll ein Popup mit einer entsprechenden Fehlermeldung ausgegeben werden, in der die beiden Reisenummern enthalten sind.
- Die Klasse soll als Funktionsobjekt (Funktor) implementiert und aufgerufen werden. Im fünften Praktikum kommen dann Konfigurationsmöglichkeiten für den Funktor hinzu.
- Das Objekt `Check` soll mit Hilfe des SIGNAL-SLOT-Mechanismus in einem `SLOT` aufgerufen werden. Das zugehörige `SIGNAL` muss über den Befehl `emit` immer dann ausgelöst werden, wenn Buchungen geändert werden.