Übungsblatt 3

Hinweise

- **Abgabe:** Bitte geben Sie Ihre Lösungen zu dieser Aufgabe digital bis zum 21.11.2021, 23:00 Uhr im Ilias ab. Spätere Abgaben werden nicht akzeptiert. Exportieren Sie Ihr Java-Projekt in *eine* zip-Datei und erstellen Sie für Ihre textuelle Antwort zu Aufgabe 3.2 ein PDF. Benennen Sie Ihre zip-Datei **blatt3_nachname_matnr.zip** und Ihr PDF **blatt3-3.2_nachname_matnr.pdf**.
- **Einzelarbeit:** Die Aufgaben zu diesem Übungsblatt sind ausschließlich in Einzelarbeit zu lösen und abzugeben; es werden keine Gruppenarbeiten akzeptiert. Bei gleichen Lösungen werden *beide* Lösungen mit 0 Punkten bewertet.
- Sie benötigen mindestens 10 Punkte, um das Übungsblatt zu bestehen. Über alle vier Übungsblätter hinweg müssen Sie mindestens 60 Punkte erreichen, d.h. im Schnitt 15 pro Blatt.

Einleitung

Laden Sie das beigefügte Java-Projekt namens *visual-mondial* in eine IDE Ihrer Wahl. Installieren Sie die Abhängigkeiten aus der Datei *pom.xml* mit *Maven*[1]. Installieren Sie als nächstes Checkstyle[2] via Maven oder installieren Sie ein Checkstyle Plug-In für Ihre IDE[3]. Nutzen Sie für Checkstyle die im Projekt enthaltene Konfigurationsdatei *google_checks.xml*. Tipp: Sie können in Ihrer IDE u.U. die Checkstyle-Regeln als Einstellungen für den Codestil importieren[4].

In den folgenden Aufgaben vervollständigen Sie das *visual-mondial* Java-Projekt mit eigenem Code. Sie dürfen dabei keine Pakete nutzen, die nicht bereits in der Datei *pom.xml* angegeben sind. Ihr Code – also alles unter dem Verzeichnis *src* – muss kompilierbar sein. Der Code muss zudem konform zu den gegebenen Checkstyle Regeln sein. Für die Abgabe exportieren Sie ihr Projekt als zip-Datei, die genau dieselbe Ordnerstruktur hat, wie die zip-Datei die sie für dieses Übungsblatt heruntergeladen haben. Fügen Sie außer Java Quellcode keine Dateien hinzu. D.h. keine Libraries, Jar-Dateien, class-Dateien, Bilddateien, etc.

Aufgabe 1 — 7 Punkte + 2 Bonuspunkte

Entwickeln Sie ein Java-Programm um Diagramme mit Statistiken zu verschiedenen Ländern zu erzeugen. Gehen Sie dabei wie folgt vor:

[1] `https://maven.apache.org/`
[2] `https://checkstyle.sourceforge.io/`
[3] z.B. `https://plugins.jetbrains.com/plugin/1065-checkstyle-idea`
[4] `http://biercoff.com/how-to-import-checkstyle-rules-into-intellij-idea-code-format-rules/`

- Erstellen Sie im Package `postgres` eine Klasse `DefaultMondialDbConnector`, die das Interface `DbConnector` implementiert. Diese Klasse soll unter dem Nutzernamen *dummy* mit dem Passwort *dummy* eine Verbindung zur Mondial-Datenbank auf *weathertop* erstellen.
- Erstellen Sie im Package `charts.country` eine Klasse `MondialCountryChartProvider`, die das Interface `CountryChartProvider` implementiert. Ihre Klasse sollte im Konstruktor einen `postgres.DbConnector` als Attribut entgegennehmen. Zudem müssen folgende Methoden implementiert werden:
  - `getLanguagesChart`: liefert ein `JFreeChart`, das alle in einem Land gesprochenen Sprachen und den Anteil der Bevölkerung, der die jeweilige Sprache spricht, zeigt.
  - `getReligionsChart`: liefert ein `JFreeChart`, das alle ethnischen Gruppen und Religionen eines Landes zeigt.
  - `getCitiesChart`: liefert ein `JFreeChart`, das die 10 größten Städte, sortiert nach Ihrer Größe, als Liste ausgibt. Um zwei Bonuspunkte zu erhalten: Heben Sie den Namen der Hauptstadt optisch hervor, wenn diese zu den 10 größten Städten gehört.
  - `update(String countryName)`: holt sich vom `postgres.DbConnector` eine Verbindung und aktualisiert damit alle Charts, die bisher über eine der `get...Chart`-Methoden erzeugt wurden. Die Charts sollen nach dem Aufruf dieser Methode ihre jeweilige Informationen über das Land anzeigen, das durch den Parameter `countryName` angegeben wird. Tipp: Um ein `JFreechart` zu aktualisieren müssen Sie lediglich die zugehörige Instanz der Klasse `Dataset` aktualisieren. Mehrere Charts können sich ein Dataset teilen.

    Stellen Sie sicher, dass alle Datenbankverbindungen direkt nach ihrer Benutzung geschlossen werden. Fangen Sie außerdem mögliche Fehler ab und geben die Fehler dann als `ChartUpdateException` an den aufrufenden Code weiter.

Testen Sie ihre Implementierung mit der Klasse `CountryChartDisplay`. Stellen Sie sicher, dass alle Datenbankverbindungen direkt nach ihrer Benutzung geschlossen werden und dass keine SQL Injections möglich sind.

Aufgabe 2 — 9 Punkte + 2 Bonuspunkte

Entwickeln Sie ein Java-Programm, welches für beliebige Datenbankanfragen die Bearbeitungsdauer durch PostgreSQL statistisch erfasst und visualisiert. Gehen Sie dabei wie folgt vor:

- Implementieren Sie in der Klasse `stats.ConfidenceInterval` zwei Methoden:
  - `forMean(double[] values, double level)` berechnet ein Konfidenzintervall für den Mittelwert einer Zufallsgröße. Dazu wird eine Stichprobe `values` genutzt. `level` ist das Konfidenzniveau des Intervalls. Berechnen Sie ein zweiseitiges Intervall für normalverteilte Zufallsgrößen mit unbekannter Varianz.
  - `forMeanDifference(double[] values1, double[] values2, double level)` berechnet ein Konfidenzintervall für die Differenz der Mittelwerte zweier Zufallsgrößen. Dazu werden zwei Stichproben `values1` und `values2` genutzt. `level` ist das Konfidenzniveau des Intervalls. Berechnen Sie ein zweiseitiges Intervall für normalverteilte Zufallsgrößen mit verschiedenen, unbekannten Varianzen.

  Ihre Methoden sollten eine *IllegalArgumentException* werfen, wenn ungültige Parameterwerte übergeben werden.

Hinweis: Nutzen Sie zur Bearbeitung dieser Aufgabe das beigefügte PDF mit Infos zum Thema Konfidenzintervalle. Tipp: In beiden Methoden müssen Sie ein Quantil der t-Statistik berechnen. Nutzen Sie dazu die Methode `inverseCumulativeProbability` der Klasse `org.apache.commons.math3.distribution.TDistribution`.

- Erstellen Sie im Package `charts.runtime` eine Klasse *PostgresRuntimesChartProvider*, die das Interface *RuntimesChartProvider* implementiert. Ihre Klasse sollte im Konstruktor einen *postgres.DbConnector* als Attribut entgegennehmen. Zudem müssen folgende Methoden implementiert werden:
  - `getRuntimesChart`: liefert ein `JFreeChart`, das die Anfragedauern zweier SQL Queries mithilfe von drei Konfidenzintervallen visuell vergleicht: ein Konfidenzintervall für die Dauer von Anfrage 1, eines für die Dauer von Anfrage 2 und eines für die Differenz der beiden Dauern.
  - `update(String query1, String query2, int numObservations, double conf)`: holt sich vom `postgres.DbConnector` eine Verbindung und aktualisiert damit alle Charts, die bisher über die `getRuntimesChart`-Methode erzeugt wurden. Dazu werden `query1` und `query2` jeweils `numObservations` mal ausgeführt und die Ausführungsdauern gespeichert. Messen Sie die Ausführungsdauern nicht selbst, da dann Störfaktoren wie Netzwerklatenzen auftreten können. Nutzen Sie stattdessen den Output des PostgreSQL Statements `EXPLAIN ANALYZE`. Nach der Messung erzeugt die Methode drei Instanzen von `ConfidenceInterval`: ein Konfidenzintervall für die Dauer von Anfrage 1, eines für die Dauer von Anfrage 2 und eines für die Differenz der beiden Dauern. Nutzen Sie dafür die zuvor implementierten Methoden `forMean` und `forMeanDifference`. Danach werden alle Diagramme aktualisiert, sodass sie diese drei Konfidenzintervalle anzeigen. Tipp: Um ein `JFreechart` zu aktualisieren müssen Sie lediglich die zugehörige Instanz der Klasse `Dataset` aktualisieren. An dieser Stelle kommt ein `IntervalCategoryDataset` in Frage.

    Stellen Sie sicher, dass alle Datenbankverbindungen direkt nach ihrer Benutzung geschlossen werden. Fangen Sie außerdem mögliche Fehler ab und geben die Fehler dann als `ChartUpdateException` an den aufrufenden Code weiter.

Testen Sie ihre Implementierung mit der Klasse `RuntimesChartDisplay`. Das Ergebnis sollte ähnlich zu Abbildung 1 aussehen. Wenn Sie eine optisch elegantere Visualisierung hinbekommen, erhalten Sie bis zu 2 Bonuspunkte.

Aufgabe 3 — 4 Punkte

Betrachten Sie Ihre Anfragen von Übungsblatt 2 genauer. Verändern Sie einzelne Anfragen syntaktisch, indem Sie beispielsweise einen Join durch eine Subquery ersetzen oder umgekehrt. Die neuen Anfragen müssen logisch äquivalent zu den Ursprungsanfragen sein. Vergleichen Sie die Ausführungsdauern der Anfragepaare mit ihrem Programm aus Aufgabe 2.

Finden Sie ein Paar aus logisch äquivalenten Anfragen, die signifikant verschiedene Dauern haben. Geben Sie das Paar an, zusammen mit einem Screenshot ihres Programms. Geben Sie zudem die Ausführungspläne an, die PostgreSQL für die beiden Anfragen erstellt. Liefern Sie angesichts der Ausführungspläne eine begründete Hypothese, wieso die eine Anfrage schneller sein könnte als die andere.

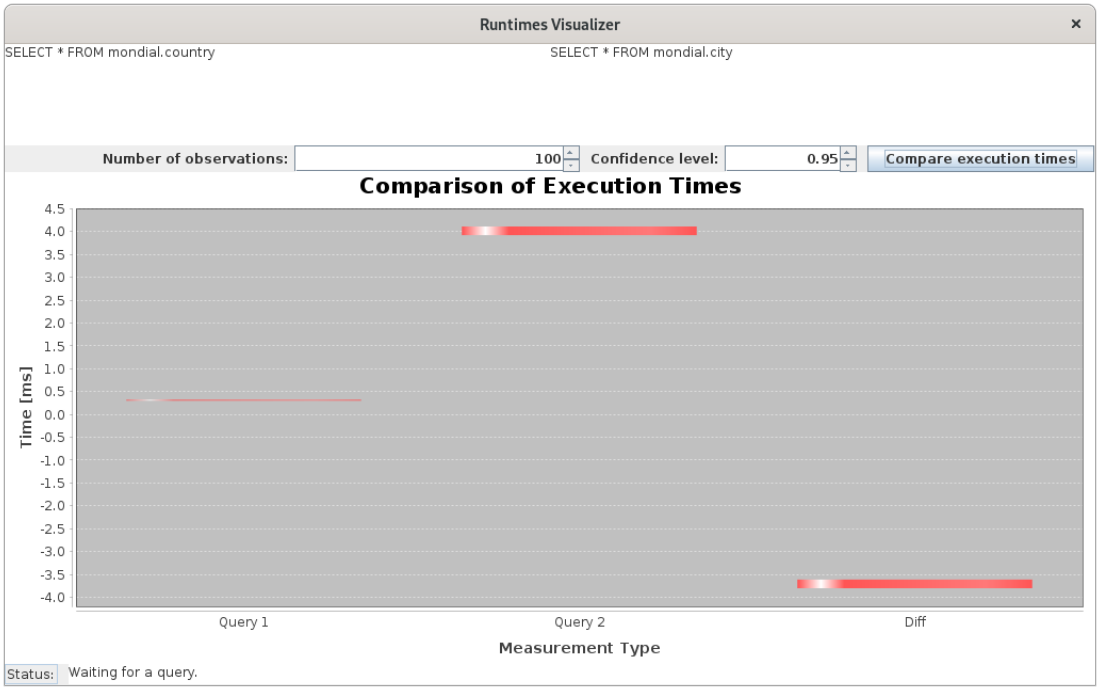

Abbildung 1: Mögliche Implementierung von Aufgabe 2.