

1 Spring

1.1 Reservations

Zu schreiben ist ein Restful Backend für eine Restaurantreservierung mit Spring. Beachten Sie die Tests. Entwickeln Sie

- Entity-Klassen kompatibel zu `data.sql`
 - Guest
 - * Auto-generated Primary-Key vom Typ `identity`
 - * String `name`
 - Table
 - * Auto-generated Primary-Key vom Typ `identity`
 - * Integer `size` - Anzahl Sitze
 - Reservation
 - * Auto-generated Primary-Key vom Typ `identity`
 - * ein Tisch wird reserviert (`non-null`)
 - * auf einen Gast (`non-null`)
 - * an einem bestimmten Datum zu einer bestimmten Uhrzeit¹ in der Zukunft
 - * für eine positive Anzahl Personen
- Rest-Endpoints
 - `get /reservations?name=guestName`
Liefert eine List aller Reservierungen auf Gäste mit Namen `guestName`
 - `get /reservations/id`
Liefert die Reservierung mit der übergebenen `id` oder `NOT_FOUND`
 - `post /reservations`
Speichert eine neue Reservierung. Dabei ist zu beachten:
 - * Status `CREATED` bzw `BAD_REQUEST`
 - * Location im Response-Header ist gesetzt
 - * Die Personenanzahl, für die reserviert wird muss kleiner gleich der `Table.size` sein.
 - * Eine Reservierung blockiert einen Tisch für 2 Stunden(exklusive der letzten Sekunde), also wenn eine Reservierung für `X` existiert, kann keine Reservierung mit Beginnzeitpunkt `(X-2):00:01 - (X+1):59:59` gespeichert werden.
- einen zentralen Exception-Handler

1.2 Space-history

Zu schreiben ist eine MVC-Webapplication mit Spring und Thymeleaf.

Im module `space-history` finden Sie einige Rohklassen. Der gegebene Code kann beliebig modifiziert werden, es ist nur **nicht** empfohlen, den `DateTimeFormatter` in `CsvParser` zu ändern. Entwickeln Sie

¹`LocalDateTime`, Zeitzonen sind zu ignorieren

- eine Entitätsklasse `Launch`, `id` beliebig
- eine Implementierung von `CsvParser`, welche die gegebene Datei `space-history.csv` in entsprechende `Launch`-Objekte parst.²
 - `Location` wird nicht verwendet
 - `Date` kann durch den gegebenen `DateTimeFormatter` gelesen werden
 - `Detail` enthält bis zum ersten `|` den Namen der Rakete, danach folgen Details über die Mission.
 - `Status Rocket` wird nicht verwendet
 - `Rocket` enthält nicht den Namen der Rakete sondern meist gar nichts oder die Schubkraft
 - `Status Mission` enthält Daten über den Erfolg der Mission. Zu unterscheiden ist einzig zwischen `Success` und `Failure`³.
 - Sie können die Datei auch parallel zur `pom.xml` ablegen, allerdings wirkt sich dies negativ auf die Beurteilung aus.
- folgende Webseiten:
 - Eine Übersicht aller Launches
 - * **absteigend** sortiert nach Datum.
 - * Der Name der Company ist ein Link zu einer ähnlichen Seite, bei der nur Missionen dieser Company aufscheinen.
 - * Mittels eines Links gelangt man zu einer Seite für das Registrieren einer neuen Mission.

Launches:

[Add new](#)

Company	Date	Rocket	Success
SpaceX	2020-08-07	Falcon 9 Block 5	success
CASC	2020-08-06	Long March 2D	success
SpaceX	2020-08-04	Starship Prototype	success
Roscosmos	2020-07-30	Proton-M/Briz-M	success
ULA	2020-07-30	Atlas V 541	success

- Eine Übersicht aller Launches **einer** Company **absteigend** sortiert nach Datum.

²Temporär bzw. falls Sie bei diesem Punkt scheitern können sie mit den Daten aus `getStartingData` arbeiten.


³Die Art des Scheiterns ist irrelevant

Launches by IAI:

Date	Rocket	Success
2020-07-06	Shavit-2	success
2016-09-13	Shavit-2	success
2014-04-09	Shavit-2	success
2010-06-22	Shavit-2	success
2007-06-10	Shavit-2	success
2004-09-06	Shavit-1	failure
2002-05-28	Shavit-1	success
1998-01-22	Shavit	failure
1995-04-05	Shavit	success
1990-04-03	Shavit	success
1988-09-19	Shavit	success

- Ein Formular zum Erfassen eines neuen Launches
 - * Company: Dropdown-Selection für alle bekannten Companies
 - * Date: Datepicker (nicht null, maximal heute), Format beliebig
 - * Rocket: Freitext, kann null sein
 - * Success: Checkbox
 - * Im Erfolgsfall redirect zur Übersicht aller Launches
 - * Im Fehlerfall ist eine entsprechende Fehlermeldung anzuzeigen und die Eingabe-seite mit den fehlerhaften Daten neu zu laden


Company: ▼

Date:  must not be null

Rocket:

Success: ☐

Company: ▼

Date:  must be a date in the past or in the present

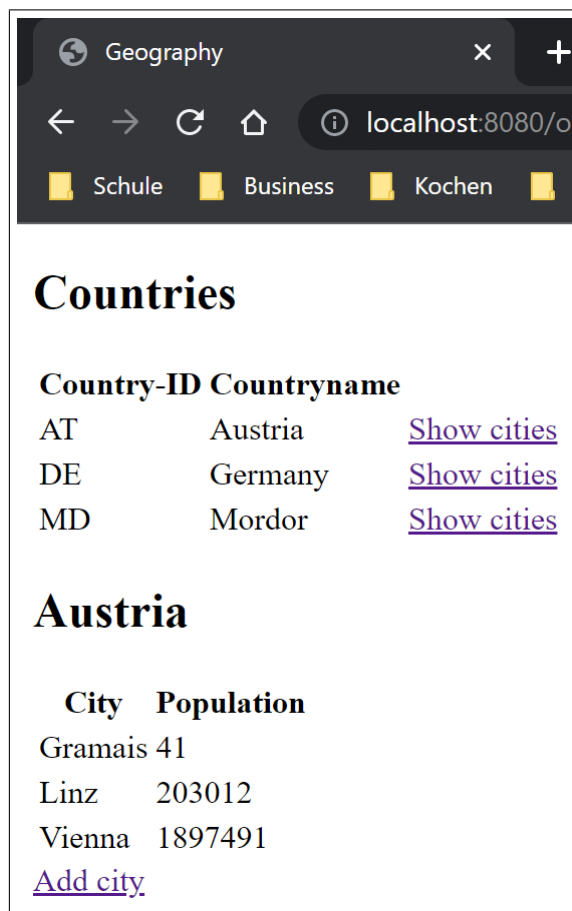
Rocket:

Success: ☒

1.3 Geography

Entwickeln Sie

- Entity-Klassen kompatibel zu `data.sql`
 - City
 - * Auto-generated Primary-Key vom Typ `identity`
 - Country
 - * Primary-Key `code` ist exakt 2 Character lang
 - * `name` beginnt mit einem Großbuchstaben
- Rest-Endpoints
 - `get /countries/{code}`
Liefert das Country mit `code` oder `NOT_FOUND`
 - `post /countries`
Speichert eine neues Country. Dabei ist zu beachten
 - * Status `CREATED` bzw `BAD_REQUEST`
 - * Location im Response-Header ist gesetzt
 - * Es darf noch kein Country mit diesem `code` existieren
- einen zentralen Exception-Handler
- folgende Webseiten:
 - Eine Übersicht aller Countries mit den entsprechenden Cities darunter
 - * `show cities` ändert den unteren Teil der Seite



- Ein Formular zum Erfassen einer neuen City

- * Im Erfolgsfall redirect zur Übersicht des jeweiligen Landes
- * Im Fehlerfall ist eine entsprechende Fehlermeldung anzuzeigen und die Eingabe-seite mit den fehlerhaften Daten neu zu laden

2 Tests

- Schreibe Tests zu den Methoden der Klasse **Service** in **testing**
- (TDD?) Implementiere **CircularBuffer** durch Anwendung des Adapter-Patterns mit einer geeigneten Collection und schreibe entsprechende Tests

3 Logik

3.1 Checkout

Zu schreiben ist Software für eine Supermarktkasse.

- **Price** - Preis eines Artikels
- **Item** - ein konkreter Artikel, welcher eingekauft wird, **ein** Apfel
`equals/hashCode` ist derartig implementiert, dass es nur auf das **Produkt** ankommt, das zum Artikel gehört
- **Product** - Artikel, welche verkauft werden, **alle** Äpfel
`equals/hashCode` ist derartig implementiert, dass es nur auf die **description** des Produkts ankommt, es gibt also nur **eine** Art von Äpfeln
- **Offer** - Angebot, x Stück eines Artikels um einen bestimmten Preis

1. Schreiben Sie in **CheckoutTest** einen **UnitTest** für die Methode **filter**.
2. Schreiben Sie nötigen Code, sodass alle Tests passen.

3.2 Orbits

In einer Simulation umkreisen Himmelskörper einander. Dies wird durch einen Graphen modelliert.

- Der Graph wird durch Strings beschrieben, wobei `a)b` bedeuten soll, dass `b` um `a` kreist.
- Der Root-Knoten sei jener, der selbst um nichts kreist.
- Wenn wir annehmen, dass alle Himmelskörper gleich weit voneinander weg sind, liefert `totalDistance` die Gesamtdistanz zum Root.

3.3 Build-Planner

Ein Projekt besteht aus mehreren Schritten, wobei viele Schritte andere voraussetzen. Der Input kommt in Form von Strings ähnlich

```
"Step B must be finished before step A can begin."  
"Step C must be finished before step A can begin."
```

- Die Klasse `BuildPlanner` bekommt derartige Strings übergeben und verarbeitet sie
- `getRequirementsForStep` retourniert eine `SortedMap`, welche als key die Schritte speichert und als value ein `Set` jener Schritte, die für den jeweiligen key Voraussetzung sind
- `order` erzeugt eine `Queue` der Schritte in der richtigen Reihenfolge
- Ungültige Strings werden ignoriert, Zyklen oder anderer unsinniger Input resultiert in einer `order-Exception`

4 Design Pattern

4.1 Censorship

Zu schreiben ist Code, um eine Zensur umzusetzen.

- Sie haben eine Applikation zum Parsen von JSON geschrieben, welche vortrefflich funktioniert, allerdings möchte ihr neuester Kunde ein neues Feature.
- Der Kunde wünscht, dass falls der JSON-String **irgendwo caseinsensitive** ein Wort aus der `forbidden.txt` enthält, das Parsing abgebrochen wird und eine `RuntimeException` geworfen wird.

```
{  
  "title": "Analysis of african soils",  
  "author": "Angie smith"  
}
```

enthält beispielsweise die verbotenen Worte "anal", "africa", "african" und "angie".
Ein einzelnes dieser Worte im übergebenen JSON reicht, dass der gesamte JSON zu verwerfen ist.

- Weil Sie mit dem Open Closed Principle vertraut sind und da die Applikation komplett ungetestet ist, entscheiden Sie sich dagegen, im vorhandenen Code Änderungen vorzunehmen, sondern ein Design Pattern zur Lösung einzusetzen.

- **Die Methode `decode` in `Decoder` darf nicht verändert werden**, alles andere schon.
1. Generieren/Zeichnen Sie ein problemangepasstes UML-Diagramm, das die Struktur des Patterns zeigt.
 2. Führen Sie die notwendigen Änderungen im Code durch.
 3. Sie müssen keine Tests schreiben, sondern können mit Hilfe der `main`-Methode testen.

4.2 Circle-shape

Sie müssen eine `awt`-Application in eine `JavaFX`-Application portieren. Dazu muss `CircleShape` implementieren. Ein Klasse `Circular` ist bereits in der Applikation vorhanden und soll weiterverwendet werden.

- Generieren/Zeichnen Sie ein problemangepasstes UML-Diagramm, das die Struktur des Patterns zeigt.
- Führen Sie die notwendigen Änderungen im Code durch.

4.3 Output

- Geben Sie das verwendete Pattern an
- Erklären Sie die Bedeutung von `OutputChannel` und `OutputFilter` im Pattern
- Geben Sie einen use-case für das Pattern an

4.4 Coffee

Sie sollen Software für eine Kaffeehauskette schreiben, welche verschiedenste Möglichkeiten für Kaffee anbietet:

- nur Kaffee
- Kaffee mit Milch(Hafer? Soja? Kuh?)
- Kaffee mit Sirup(Karamel? Vanille?)
- Kaffee mit Zucker
- Erstellen Sie Klassen, welche beliebige Kombinationen ermöglichen. Für jedes derartige Produkt soll ein Gesamtpreis und eine Liste der Zutaten abgerufen werden können.
- Schreiben Sie einige Tests für ihre Application.

4.5 Coffee

Schreiben Sie eine Klasse `Exporter`, die je nach Bedarf `xml` oder `pdf`-Files aus einem `Document` erzeugt. Schreiben Sie eine Application, die einen `Exporter` anlegt und mit dem selben Objekt einmal `pdf` und einmal `xml` erzeugt.

4.6 Soccer

- Skizzieren Sie einen alternativen Lösungsansatz
- Geben Sie das verwendete Pattern an
- Implementieren Sie die Klasse `SoccerTeam`

4.7 Mystery

- Geben Sie das verwendete Pattern an
- Beschreiben Sie den use-case des Design Patterns und die Rollen von **Mystery** und **Other** (umbenennen!)
- Implementieren Sie das Pattern vollständig, achten Sie auf SOLID-Konformität
- Testen Sie ihre Implementierung mit einer geeigneten Application oder durch Tests

4.8 Transactions

Sie finden eine Klasse **Account** vor, welche ein Konto repräsentiert. IBANs werden der Einfachheit halber als String gespeichert und müssen **nicht** validiert werden.⁴

- Ergänzen Sie die Methoden gemäß den Javadocs
- Schreiben Sie Tests zu den Methoden
- Machen Sie **Account** im Sinne des Observer-Patterns observable. Bei der Registrierung eines **Observers** ist eine IBAN anzugeben und für **jede** Transaktion die auf diesem Konto stattfindet ist zu informieren.
- Implementieren Sie einen **Observer**, welcher auf der Konsole für jede Transaktion ausgibt, ob sie erfolgreich war oder nicht, den jeweiligen **amount** sowie einen Timestamp⁵.

4.9 Primes

Sie finden eine Klasse **PrimeFinder** vor, welche Primzahlen findet.

- Die Klasse wird in einer Application verwendet, welche möglicherweise für dieselbe Primzahl mehrmals aufgerufen wird
- Schreiben Sie einen Cache, welcher für bereits berechnete Werte die Primalität speichert, sodass für jede Zahl nur einmal berechnet wird, ob diese prim ist oder nicht
- Beim Instanzieren des Caches soll dieser bereits für alle Zahlen bis 10000 gefüllt sein

⁴Jeder String außer **null** ist als valid zu werten.

⁵**Instant.now()**