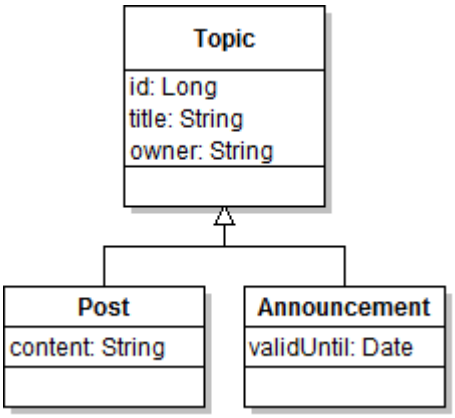
Arbeitsblatt JPA: Inheritance

Ziele
Ziel dieses Arbeitsblattes ist es, dass sie die Auswirkung unterschiedlicher Inheritance-Strategien verstehen.



Ausgangslage
Im Gitlab-Projekt https://gitlab.fhnw.ch/eaf/hs21/04 steht das Projekt lab-jpa-inheritance zur Verfügung, welches für diese Aufgaben verwendet werden soll. Importieren Sie dieses Projekt in ihre IDE.

Wie im Arbeitsblatt von letzter Woche verwenden wir auch in diesem Projekt Spring um den Entity-Manager zu erzeugen. Die Parameter in der Datei `application.properties` sind so gesetzt, dass eine in-memory Datenbank verwendet wird:
`spring.datasource.url=jdbc:h2:mem:lab-jpa-db`

Aufgaben:

1) Im Projekt werden die in der Figur gezeigten drei Entitäten definiert. Ändern Sie die Inheritance-Strategie von `SINGLE_TABLE` (dem Default) auf `JOINED` und auf `TABLE_PER_CLASS`, in dem Sie bei der abstrakten Basisklasse `Topic` die Annotation

   ```
   @Inheritance(strategy=InheritanceType.JOINED)
   ```

   bzw.

   ```
   @Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
   ```

   hinzufügen. Starten Sie danach das Programm `Test`, welches zwei Entitäten in der Datenbank ablegt, je eine Instanz der Klassen `Post` und `Announcement`. Schauen Sie sich mit Hilfe der H2-Konsole das generierte Schema an (dieses können Sie bereits aus dem Log-Output ableiten) und prüfen Sie, wie die Daten der beiden Entitäten in den Tabellen abgelegt werden.

2) Die Post-Instanz, welche im Test-Programm eingefügt wird, hat keinen Content. Ändern sie die Definition der Entität `Post` so ab, dass das Feld `content` nicht mehr optional ist; dies kann mit der `@Basic`-Annotation erreicht werden:

   ```
   @Basic(optional=false)
   private String content;
   ```

   Beim Start der Applikation sollten Sie nun eine `ConstraintViolationException` sehen (da das content-Feld im Testprogramm nicht gesetzt wird), und zwar sowohl für die Vererbungsstrategie `JOINED` als auch für `TABLE_PER_CLASS`.

3) Wechseln Sie nun wieder zurück auf die Vererbungsstrategie `SINGLE_TABLE`, behalten Sie jedoch die unter 2) gemachte Änderung dass das Feld content nicht mehr optional ist bei, und führen Sie das Testprogramm, welches einen Post ohne Content einfügt, nochmals aus. Wird der erwartete Fehler geworfen?

4) In diesem Schritt definieren wir das Schema von Hand und fügen einen Constraint hinzu. Ändern Sie dazu die Datei `application.properties` wie folgt ab:

```
spring.jpa.hibernate.ddl-auto=none
spring.sql.init.mode=always
```

und fügen Sie in der Datei `schema.sql` folgenden SQL CHECK hinzu:

```
ALTER TABLE Topic ADD CONSTRAINT post_content_check CHECK (
       CASE WHEN DTYPE = 'Post'
       THEN CASE WHEN CONTENT IS NOT NULL THEN 1 ELSE 0 END
       ELSE 1
       END = 1
);
```

Wenn Sie jetzt das Testprogramm wieder starten, dann sollten Sie wieder eine `ConstraintViolationException` sehen. Wenn Sie nun dem im Test generierten Post einen Inhalt mitgeben, dann sollte die Exception nicht mehr geworfen werden.

5) Für den nächsten Task müssen Sie die für Task 4) gemachten Änderungen in der Datei `application.properties` wieder rückgängig machen, d.h. das Schema soll wieder von JPA/Hibernate erzeugt werden:

```
spring.jpa.hibernate.ddl-auto=create-drop
spring.sql.init.mode=never
```

(Dies kann auch mit einem „git restore“ erreicht werden).

Ändern Sie nun die PK-Strategie auf Auto-Increment in dem Sie beim `id`-Feld in der Klasse `Topic` folgende Deklaration hinzufügen:

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

und prüfen Sie, welche Vererbungsstrategien nun noch verwendet werden können.

6) Probieren Sie als nächstes neben `test1()` auch noch `test2()` aus, der alle Topics aus der Datenbank liest. Es handelt sich dabei um ein polymorphes Query.