Ćwiczenie: Modelowanie Aggregate i Value Objects

Rozszerz klasę **Device** pozwalającą docelowo konfigurować informacji o urządzeniu takich jak:
- lokalizacja urządzenia
- godziny dostępności urządzenia
- ogólne ustawienia
- oraz pozwala na przypisanie urządzenia do operatora urządzeń (**operator**) i dostawcy usługi (**provider**).

Ponad to klasa docelowo udostępnia informacje o aktualnych brakach i błędach w konfiguracji oraz o widoczności urządzenia dla klientów końcowych.

UWAGA agregat demonstrowany w tym ćwiczeniu to typowy model definicyjny (obiekty typu draft) i posiada następujące cechy:
- relatywnie prosta logika CRUD-owa, metody typu get / set
- miękka weryfikacja reguł podczas edycji:
  - złamanie reguł jest dopuszczalne
  - posiada ciągłą informację o aktualnych błędach, żądaniach lub sugestiach naprawy
- złamane reguły mogą blokować pewne funkcje, tutaj publikację / widoczność urządzenia dla klientów

Zapoznaj się z obecnym stanem klasy **Device** oraz **OpeningHours** i **Settings** oraz dodaj kolejne konfigurowalne informacje o urządzeniu**:**

1. Zaimplementuj klasę **Location** z polami tekstowymi
   street, houseNumber, city, postalCode, state, country
   oraz coordinates, gdzie **Coordinates** to klasa o polach longitude, latitude typu BigDecimal
2. Do klasy **Device** dodaj metodę updateLocation oraz pole location
3. Zaimplementuj klasę **Ownership** z polami tekstowymi operator i provider
4. Do klasy Device dodaj metodę assign(**Ownership**) oraz pole ownership

Dodaj weryfikację braków i błędów informacji oraz kalkulację widoczność urządzenia:

1. Zaimplementuj metodę getViolations zwracającą nową klasę **Violations** konstruowaną za pomocą wzorca budowniczy
2. **Violations** niech posiada serię pól typu Boolean dla każdego weryfikowanego błędu:
   - operatorNotAssingned
   - providerNotAssigned
   - locationMissing
   - showOnMapButMissingLocation
   - showOnMapButNoPublicAccess
3. W metodzie getViolations klasy **Device**, która przy każdym jej wywołaniu wylicz wszystkie weryfikowane błędy
4. Zaimplementuj klasę **Visibility** z polami forCustomer oraz roamingEnabled
   forCustomer to enum o 3 wartościach: USABLE_AND_VISIBLE_ON_MAP, USABLE_BUT_HIDDEN_ON_MAP, INACCESSIBLE_AND_HIDDEN_ON_MAP
   roamingEnabled to pole typu boolean
5. Do klasy **Device** dodaj metodę getVisibility zwracającą obiekt typu **Visibility**, która przy każdym jej wywołaniu wylicz widoczność według poniższych reguł:
   1. roamingEnabled = true kiedy nie ma błędów i braków w danych oraz settings.publicAccess == true
   2. analogicznie reguły muszą być spełniony by klient mógł użyć urządzenia
      część USABLE / INACCESSIBLE wartości enuma ForCustomer
      ponadto settings.showOnMap decyduje czy urządzenie jest pokazywane na mapie
      część VISIBLE_ON_MAP / HIDDEN_ON_MAP wartości enuma ForCustomer

Ćwiczenie: Persystencja obiektu jako seria zdarzeń

Rozszerz klasę **Device** by konstruowała zdarzenia przy każdej modyfikacji swojego stanu, następnie każde zdarzenie zapisz w bazie danych. Ponad to zaimplementuj klasę **DeviceRepository,** która wczyta z bazy danych istotne zdarzenia i skonstruuje klasę **Device**:

1. W każdej metodzie klasy **Device**, modyfikującej stan urządzenia, skonstruuj odpowiednie zdarzenie mówiące o zmianie poszczególnych informacji. np: w metodzie updateLocation skonstruuj zdarzenie **LocationUpadated** posiadające id urządzenia oraz nową lokację
2. Do klasy **Device** dodaj pole package scope List<DomainEvent> events, gdzie DomainEvent to marker interfejs dla wszystkich typów zdarzeń
3. Zaimplementuj klasę **DeviceRepository** posiadającą dwie metody:
   - Device get(deviceId)
   - void save(Device)
4. Zaimplementuj encję **DeviceEventEntity** oraz repozytorium Spring Data JPA **DeviceEventRepository**.

```
@Data
@Entity
@Table(name = "device_events")
class DeviceEventEntity {
    @Id
    private UUID id;
    private String deviceId;
    private Instant time;
    @Type(type = "jsonb")
    @Column(columnDefinition = "jsonb")
    private DomainEvent event;
}
```

Persystencja obiektu jako JSON w kolumnie typu binary json w bazie Postgresql

5. By Jackson potrafił odczytywać polimorficzne obiekty event-ów dodaj poniższe adnotacje do interfejsu **DomainEvnet:**

```
@JsonTypeInfo(use = JsonTypeInfo.Id.NAME)
@JsonSubTypes({
        @JsonSubTypes.Type(value = LocationUpdated.class, name = "LocationUpdated"),
        ...
```

6. W metodzie save **DeviceRepository** wyjmij z instancji klasy Device listę wyemitowanych event-ów i zapisz każdy z event-ów w bazie.
7. W metodzie get **DeviceRepository** pobierz z bazy wszystkie eventy dla danego deviceId pogrupuj je po typie i wybierz ostatni event z każdego typu, pamiętaj że pewne eventy mogły nie występować inne wielokrotnie. Grupowanie „ostatnich” event-ów po stronie Postgresa:

```
@Query(value = "select distinct on (event ->> '@type') *" +
        " from device_events" +
        " where deviceId = :deviceId" +
        " order by payload ->> '@type', time desc", nativeQuery = true)
List<DeviceEventEntity> findLastEvents(String deviceId);
```

8. Skonstruuj instancję Device przekazując do konstruktora wybrane wartości z ostatnich event-ów. Możesz też przekazać przez konstruktor całą listę event-ów lub posłużyć się fabryką czy budowniczym.