JPA - cykl życia encji

Entity

➔ zwykła klasa Javy **POJO** (ang. **P**lain **O**ld **J**ava **O**bject) + informacje potrzebne do mapowania do tabeli w bazie danych (XML lub adnotacje)

javax.persistence.EntityManager

➔ interfejs z metodami do pracy z bazą danych (**CRUD**, złożone zapytania, transakcje):
- *perist()* - zapisz encję w bazie danych
- *remove()* - usuń encję z bazy
- *merge()* - przywróć/zsynchronizuj encję do aktualnego **Persistence Context**
- *find()* - znajdź pojedynczą encję po podanym id
- *createQuery().getResultList()* - pobierz listę encji

➔ ściśle powiązany z **Persistence Context**
➔ **NO! try-with-resources** - niestety nie działa w tym przypadku!
➔ zamykamy ręcznie przez metodę *close()* albo zamykając fabrykę z której powstał

➔ **Persistence Context:**
- cache (**first-level cache**) przechowujący encje zarządzane przez JPA w danej sesji
- bufor pomiędzy encjami, a bazą danych. Znajdują się tam wszystkie encje, które zostały pobrane lub zapisane w bazie danych w ramach pojedynczej sesji
- zaczyna się wraz ze stworzeniem obiektu ***EntityManager*** i kończy się w momencie zamknięcia ***EntityManager*** (metoda *close()*)
- wyznacza zakres pojedynczej sesji logicznej

➔ **sesja/logiczna transakcja**
- zestaw operacji które wykonują konkretne wymaganie biznesowe (logikę biznesową)
- zwykle jest to jedna metoda w klasie typu **DAO**
- może objąć kilka transakcji bazodanowych.

➔ **transakcja(fizyczna, bazodanowa)** -
- zestaw operacji na bazie danych
- traktowane są jak jedna operacja
- zasada: wszystko-albo-nic

Autor: Jarosław Skarzyński

javax.persistence.EntityManagerFactory

➔ fabryka służąca do tworzenia obiektów klasy ***EntityManager*** dla konkretnego **Persistence Unit**
➔ tworzymy raz na każdą bazę danych używaną w aplikacji

➔ **Persistence Unit:**
- ◆ wszystkie klasy-encje zgrupowane w jeden zbiór
- ◆ konfiguracja bazy danych
- ◆ parametry dodatkowe (np.: show_sql, hbm2ddl.auto)

Zadanie nr 7:

1. Przejdź do modułu **jpa-starter** i sprawdź konfigurację bazy danych w pliku **persistence.xml** dla Persistence Unit o nazwie: **pl.sda.jpa.starter.lifecycle.** Powinna wskazywać na bazę: **jpa_test**

2. Znajdź klasę ***JpaLifeCycle***. Uruchom metodę *persistentContextLifeCycleTest()* i sprawdź czy działa. Przeanalizuj krok po kroku linie kodu.

3. W **persistence.xml** dla PU podanym w pkt 1 zmień parametry tak żeby Hibernate nie generował tabel bazy danych automatycznie

4. Stwórz klasę ***JpaManager*** w pakiecie: **pl.sda.jpa.starter.lifecycle.** W metodzie *main()* klasy ***JpaManager*** stwórz instancję klasy ***EntityManagerFactory*** (zobacz jak to jest zrobione w klasie ***JpaLifeCycle***) i przekaż ją do klasy ***CourseEntityDao*** (jako parametr konstruktora). W klasie ***CourseEntityDao*** zapisz instancję ***EntityManagerFactory*** jako pole i użyj jej do tworzenia obiektów ***EntityManager*** w metodach.

5. Otwórz klasę ***CourseEntityDao*** i uzupełnij metody kodem z JPA. Sprawdź czy działają testując je w klasie ***EntityManager***, np.:
   a. dodaj kilka encji CourseEntity do bazy
   b. pobierz encję CourseEntity o podanym id z bazy i usuń ją
   c. pobierz wszystkie encje, usuń ostatnią z listy.
   d. pobierz jedną encję z bazy po id, zmień jej nazwę i datę zakończenia kursu, zaktualizuj dane w bazie

6. **(dla chętnych)** Stwórz w klasie ***CourseEntityDao*** metodę która dla wszystkich kursów zaczynających się w przedziale czasu podanym jako parametry metodu (np. *changeSchedule(Date start, Date end)*) przesunie datę startu i końca kursu o miesiąc do przodu. Metoda powinna zwrócić wszystkie encje które zostały zmienione