Hibernate

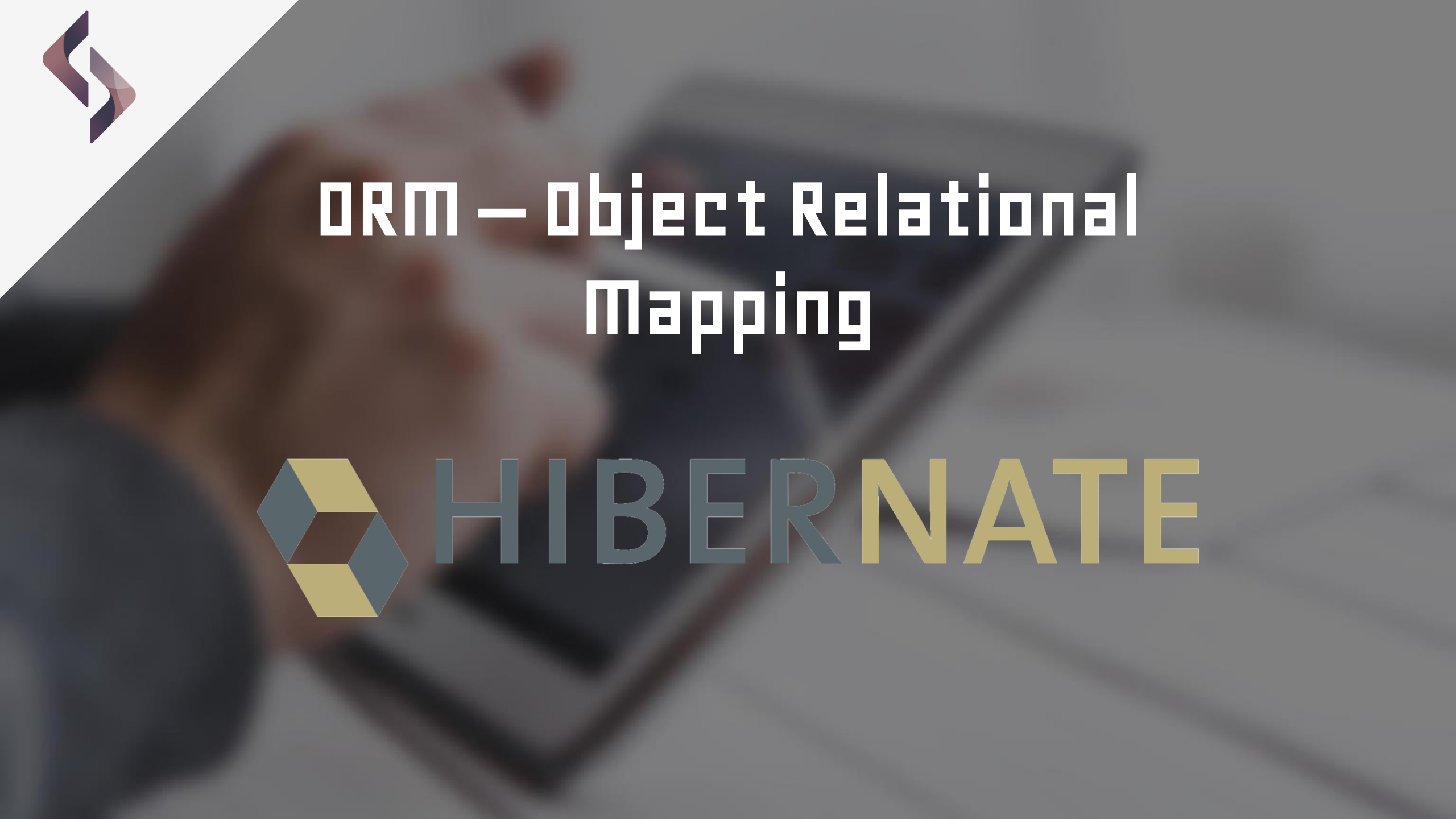
Adam Łagoda

Agenda

1. Czym jest Hibernate
2. Konfiguracja w xml-u
3. Konfiguracja adnotacjami
4. CRUD
5. JPA –cykl życia encji
6. JPA – mapowanie relacji
7. JPA - JPQL

Przydatne informacje

1. Repozytorium jest dostępne pod https://github.com/adamlagoda/javadb-starter
2. Rozwiązania zadań znajdują się na branchach nazywanych wg schematu ***zadanie[NR_ZADANIA]_solution*** np. *zadanie2_solution*
3. Rozwiązania zadanań są niezależne od poprzednich, tzn. program może nie uruchomić się prawidłowo, gdy przełączymy się z *master* na branch z rozwiązaniem.
4. Prezentacja jest dostępna w formie PDF-a w repozytorium

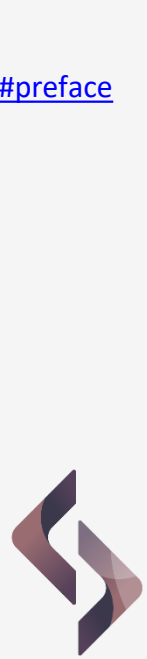


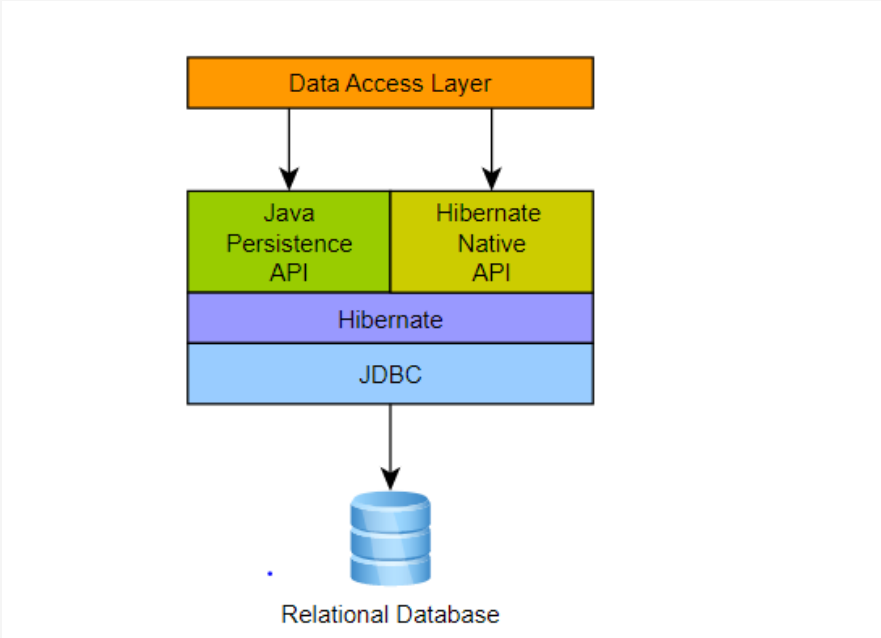
ORM – Object Relational Mapping

HIBERNATE

Hibernate

- Model Obiektowy ⇔ Model Relacyjny,
- Oszczędza ok 95% kodu
  https://docs.jboss.org/hibernate/orm/5.3/quickstart/html_single/#preface
- Do aplikacji data-oriented raczej nieprzydatny,
- Dane w postaci tabeli ⇔ graf,
- Istnieją inne implementacje ORM, np. EclipseLink





Konfiguracja w xml-u

- Hibernate składa się z podzbiorów bibliotek:
  - `hibernate-core` – API i zapytań i implementacja ORM
  - `hibernate-hikaricp` – implementacja poolingu
  - `hibernate-ecache` - implementacja cache'a
  - Itd..
- `hibernate.cfg.xml` – plik konfiguracyjny,
- Encja musi mieć bezargumentowy konstruktor,
- Mapping w pliku `[nazwa_pliku].hbm.xml`,
- Typy mapowane wg szablonu Hibernate, nie JDBC,

Konfiguracja

Hibernate API:

org.hibernate.cfg.Configuration (1 sposób konfigurowania Hibernate, przechowuje konfigurację i mapowania w jednej klasie Configuration, z której można zbudować SessionFactory)
org.hibernate.boot.registry.StandardServiceRegistry (2 sposób konfigurowania Hibernate, przechowuje konfigurację Hibernate tj.: dostęp do bazy danych, parametry, cache itp)
org.hibernate.boot.MetadataSources (2 sposób konfigurowania Hibernate, przechowuje informacje o mapowaniach klas-encji (XML, adnotacje), tworzy obiekt Metadata

Zadanie 0

1. Stwórz nową lokalną bazę danych w MySQL. Nazwij ją: `hibernate_test`. Otwórz plik `hibernate.cfg.xml` i zmień ustawienia bazy danych tak żeby wskazywały na bazę, którą przed chwilą stworzyłeś.
2. Znajdź klasę `HibernateConfiguration`, uruchom i sprawdź, czy działa. Sprawdź czy w nowej bazie danych pojawiła się tabela: `courses` i czy ma dane.
3. * Spróbuj dodać do mapowania (`hibernate.cfg.xml`) klasę `Student` (należy stworzyć osobny plik z mapowaniem: `Student.hbm.xml`). Utwórz nową instancję klasy Student w `HibernateConfiguration` i zapisz ją w bazie. Sprawdź, czy pojawiła się nowa tabela z danymi.

Zadanie 1

1. Otwórz plik `hibernate.cfg.xml` i zakomentuj linię: `<mapping resource="Course.hbm.xml"/>`. Odkomentuj linię: `<mappig class=„org.example.hibernate.starter.entity.CourseEntity"/>`
2. Znajdź klasę `HibernateNaConfiguration`, uruchom i sprawdź czy działa. Sprawdź czy w nowej bazie danych pojawiła się tabela: `courses` i czy ma dane.
3. Zmień ustawienia Hibernate za pomocą metod w `StandardServiceRegistryBuilder` - nadpisując parametry (np.: `'show_sql'`) z pliku `hibernate-na.cfg.xml`, uruchom program i sprawdź czy zmiana działa.
4. Stwórz nową klasę `StudentEntity` w pakiecie `org.example.hibernate.starter.entity`, możesz skorzystać z informacji umieszczonych w klasie: `org.example.hibernate.starter.entity.CourseEntity`
5. Dodaj klasę `StudentEntity` do konfiguracji Hibernate przez wywołanie odpowiedniej metody klasy `MetadataSources`, następnie stwórz nową instancję klasy i zapisz ją w bazie


CRUD

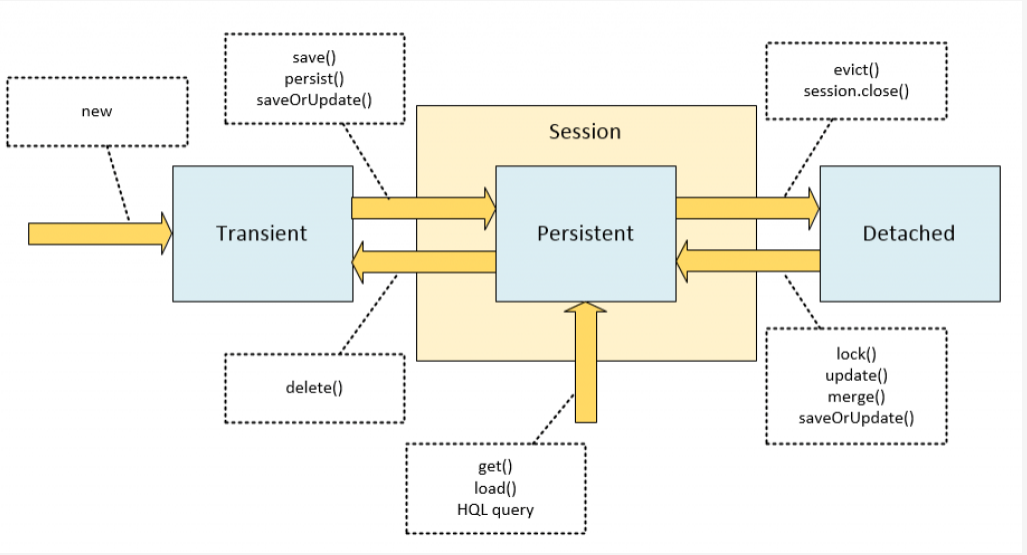
CRUD

Hibernate API:

org.hibernate.SessionFactory
(fabryka służąca do tworzenia obiektów klasy Session, przechowuje dane o mapowaniach ORM,"ciężki" obiekt, zwykle tworzy się jedną instancję na każdą bazę danych używaną w aplikacji)
org.hibernate.Session
(główna klasa Hibernate dostarczająca API do pracy z bazą danych (CRUD, zapytania, transakcje), "lekki" obiekt, tworzymy go za każdym razem gdy potrzebujemy interakcji z bazą danych, zawsze zamykamy na koniec!)

CRUD

- **Transakcja(fizyczna, bazodanowa)** - zestaw operacji na bazie danych, które traktowane są jak jedna operacja. Albo są wdrożone wszystkie operacje albo żadna.
- **Sesja/logiczna transakcja** - zestaw operacji/zadań które wykonują konkretne wymaganie biznesowe (logikę biznesową) określoną w aplikacji. Transakcja logiczna może objąć kilka transakcji bazodanowych. Cykl życia obiektu `Session` jest związany z początkiem i końcem sesji logicznej.
- Obiekt typu `Session` reprezentuje *persistence context*



Zadanie 2

1. Znajdź klasę `HibernateLifeCycleTest` i uruchom testy . Zwróć uwagę na kiedy Hibernate wywołuje zapytania oraz sprawdź stan bazy po uruchomieniu każdego testu. Sprawdź, czy stan bazy zmieni się, gdy pominiemy fragmenty `session.beginTransaction()` i `transaction.commit()`.
2. Uzupełnij metody w klasie `CourseEntityDao`, które wykonają operacje CRUD dla encji: `CourseEntity`. Przetestuj działanie funkcji w metodzie `main`.
3. Stwórz klasę `StudentsEntityDao` i stwórz dla niej operacje CRUD opartą o Hibernate.

JPA – cykl życia

JPA – cykl życia

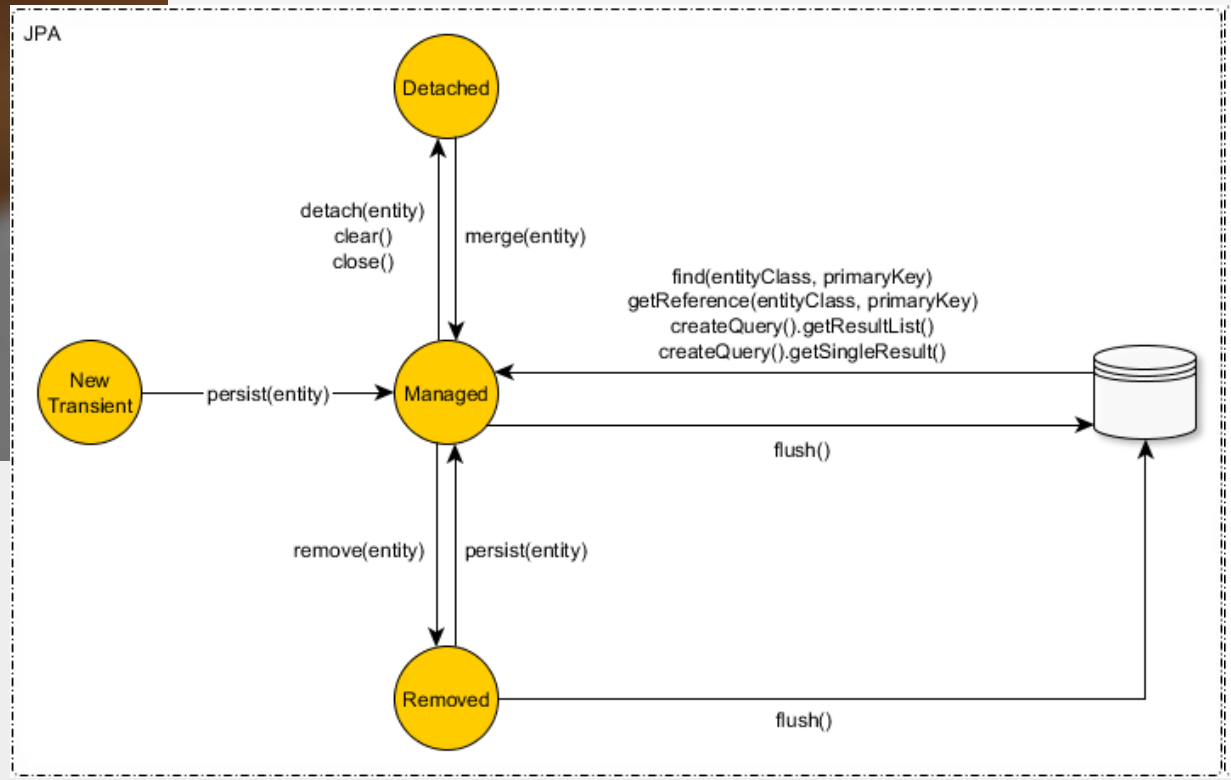
JPA API:

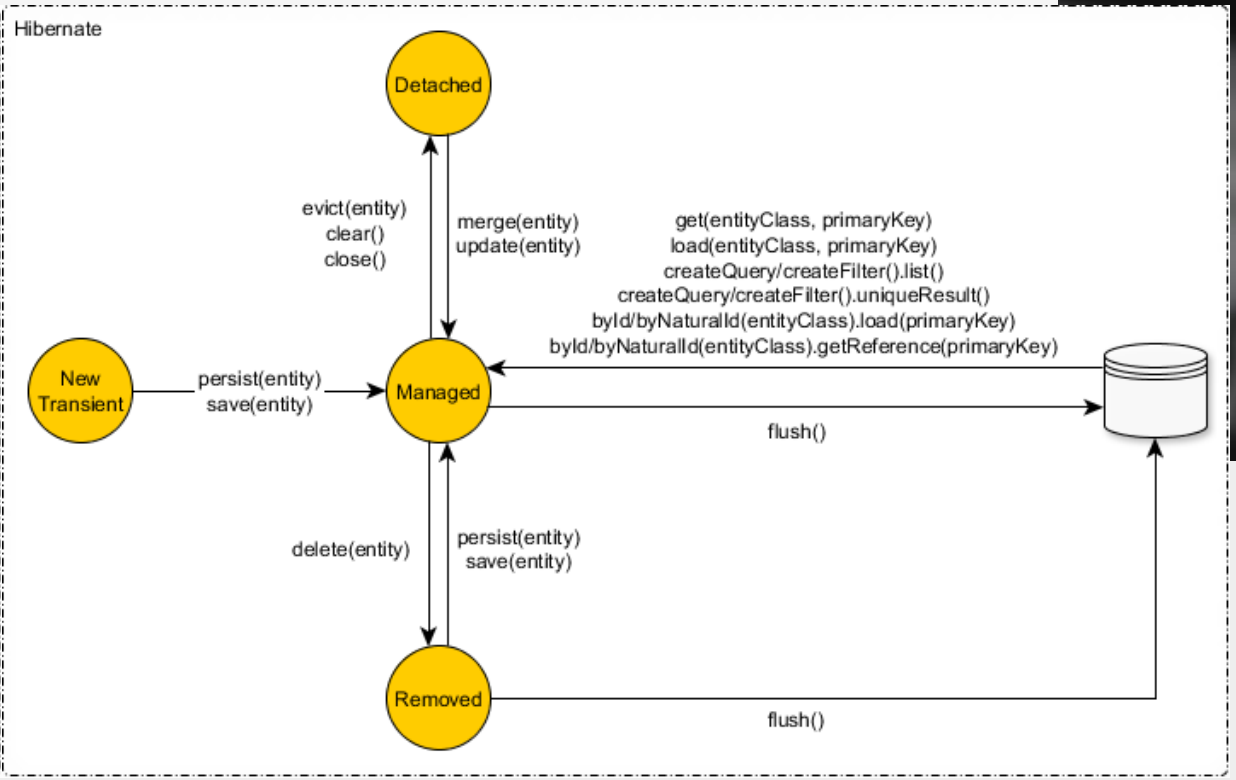
javax.persistence.EntityManagerFactory
(fabryka służąca do tworzenia obiektów klasy EntityManager dla konkretengo Persistence Unit ,"ciężki" obiekt, zwykle tworzy się jedną instancję na każdą bazę danych używaną w aplikacji)
javax.persistence.EntityManager
(interfejs do interakcji z konkretnym Persistence Unit, zawiera metody do operacji CRUD, obsługi transakcji, a także złożonych zapytań)

JPA – cykl życia

- **JPA** - (ang . ***J**ava **P**ersistence **API)*** - specyfikacja (JSR 338) opisująca standardowe podejście platformy Java do technologii ORM. Hibernate implementuje standard JPA. Jest to część platformy Java EE/Jakarta EE.
- **Persistence Unit** - wszystkie klasy-encje zgrupowane w jeden zbiór przez aplikację, które są mapowane do pojedynczej bazy danych.
- **Persistence Context** - cache (*first-level cache*) przechowujący encje zarządzane przez JPA w danej sesji. PC to rodzaj bufora pomiędzy encjami, a bazą danych. Znajdują się tam wszystkie encje, które zostały pobrane lub zapisane w bazie danych w ramach pojedynczej sesji. PC w swojej podstawowej formie trwa tak długo jak długo "żyje" obiekt `EntityManager` z nim związany (od utworzenia instancji `EntityManager` do wywołania metody `close()`) .
- W Hibernate Persistence Context jest reprezentowany przez `Session`. Gdy z kolei korzystamy z Hibernate'a, jako implementacji JPA, to `EntityManager` „opakowuje" obiekt `Session`

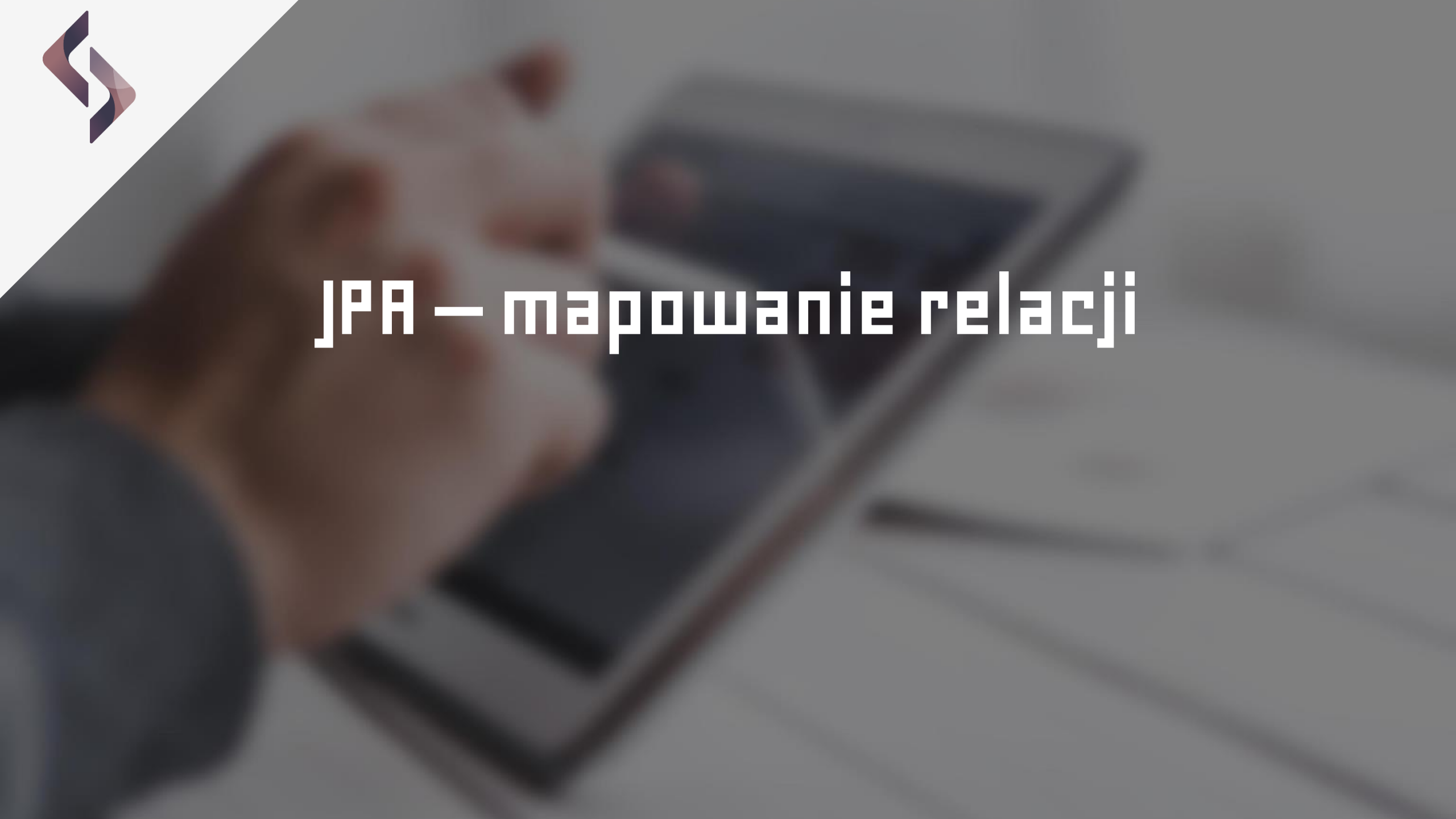
JPA – cykl życia





Zadanie 3

1. Przejdź do modułu `jpa-starter` i zmień konfigurację bazy danych w pliku `resources/META-INF/persistence.xml` tak żeby wskazywała na bazę: `hibernate_test`
2. Znajdź klasę `JpaLifeCycle`. Uruchom metodę `main` i przeanalizuj kod.
3. W pliku `META-INF/persistence.xml` zmień ustawienie parametru: '`hibernate.hbm2ddl.auto`' na '`create-drop`'.
4. Otwórz klasę `JpaTest` i w metodzie `main` dodaj kod, który skonfiguruje i stworzy obiekt `EntityManagerFactory` (zobacz jak to jest zrobione w klasie `JpaLifeCycle`). Następnie poeksperymentuj z różnymi metodami obiektu `EntityManager`, np.:
    a. dodaj kilka encji `CourseEntity` do bazy,
    b. pobierz encję `CourseEntity` o podanym id z bazy i usuń ją,
    c. pobierz wszystkie encję za pomocą `createQuery(.getResultList()`, usuń ostatnią z listy,
    d. pobierz jedną encję z bazy po id, zmień jej nazwę i datę zakończenia kursu, zaktualizuj dane w bazie
6. Uzupełnij metody z klasy `CourseEntityDao` za pomocą kodu JPA. Wykorzystaj metody: (`persist()`, `merge()`, `delete()`, `createQuery()`)



JPA – mapowanie relacji

JPA – mapowanie relacji

- **Rodzaje relacji:**
  1. **One-to-One** (`@OneToOne`) - relacja jeden-do-jednego
  2. **Many-to-One** (`@OneToMany`, `@ManyToOne`) - relacja jeden-do-wielu
  3. **Many-to-Many** (`@ManyToMany`) - relacja wiele do wielu
- **Kierunek relacji (dotyczy tylko Javy):**
  1. **Unidirectional** - jednokierunkowy
  2. **Bidirectional** - dwukierunkowy
- **Właściciel relacji (**`mappedBy = "field_name"`**)** - jedna klas w relacji dwukierunkowej powinna być wskazana jako właściciel relacji, przez atrybut `mappedBy`. Dzięki temu JPA “wie” gdzie szukać danych do relacji.

Zadanie 4

1. Stwórz bazę danych `jpa_relations_test`
2. W pliku `META-INF/persistence.xml` zmień ustawienie parametru: ‘`hibernate.hbm2ddl.auto`’ na ‘`create`’
3. Przejdź do klasy `JpaRelations` i uruchom metodę `oneToOne()`. Przeanalizuj kod.
4. Dodaj do `StudentEntity` nową relację typu one-to-one przenosząc do osobnej klasy (`SeatEntity`) informacje z pola seat. Dodaj odpowiednie adnotacje i sprawdź czy relacja działa. Nowa klasa powinna mieć trzy pola (poza id):
   a. `String columnNumber` - wartość kolumny, np.: ‘A’, ‘B’ …
   b. `int rowNumber` - numer rzędu, np.: 1, 2, 3
   c. `Int seatNumber` - numer siedzenia, np.: 1, 2
4. Przejdź do klasy `JpaRelations` i uruchom metodę `oneToMany()`. Przeanalizuj kod. Następnie:
   a. stwórz nowy kurs, dodaj kilku studentów do kursu i zapisz zmiany w bazie. Zapisz tylko studentów i sprawdź czy kurs również został dodany.
   b. wyciągnij kurs, sprawdź czy ma studentów, usuń studenta o id: 1, zmień imię studenta o id:2 i dodaj nowego studenta do kursu. Sprawdź czy zmiany zapisały się w bazie.

Zadanie 4

5. Przejdź do klasy `JpaRelations` i uruchom metodę `manyToMany()`. Przeanalizuj kod. Następnie:
   a. spróbuj dodać parę nowych skilli i przypisz je do któregoś ze studentów
   b. usuń tego studenta i sprawdź czy usunięta została relacja między studentem i skillem
   c. * ustaw relację student-skill jako dwukierunkową. Właścicielem relacji ma być obiekt `SkillEntity`
   d. * wyciągnij jakiś skill z bazy i sprawdź jacy studenci są do niego dopisani.
6. * Stwórz nową encję `CoachEntity` (z polami: `id, name`). Dodaj relację many-to-many między trenerami a kursami. Relacja powinna być dwukierunkowa. Przetestuj swój kod:
   a. dodaj kilku trenerów do dwóch różnych kursów
   b. wyciągnij trenerów z kursu pierwszego i usuń relację między jednym z trenerów a konkretnym kursem
   c. wyciągnij wszystkich trenerów z danego kursu
   d. wyciągnij wszystkie kursy do których przypisany jest dany trener


JPA - JPQL

JPA - JPQL

- **JPQL** (ang. ***J**ava **P**ersistence **Q**uery **L**anguage)* - to niezależny od bazy danych zorientowany obiektowo język zapytań zdefiniowany w ramach specyfikacji JPA. Używa się go do wykonywania zapytań do bazy danych, bardzo przypomina język SQL ale operuje na encjach JPA (obiektach), a nie na tabelach bazy danych. Wspierane zapytania to: `SELECT`, `UPDATE`, `DELETE`.
- **HQL** (ang. ***H**ibernate **Q**uery **L**anguage)* - język zapytań biblioteki Hibernate. Stworzony wcześniej niż JPQL. JPQL był mocno inspirowany podczas tworzenia specyfikacją HQL.

Zadanie 5

1. Dodaj do konfiguracji JPA (Persistence Unit) - wszystkie klasy z pakietu `entities`. Upewnij się że wszystkie relacje w tych klasach działają.
2. Znajdź klasę `JpaQueries`. Uruchom metody `simpleQuery(), relationsQuery()` i przeanalizuj krok po kroku linie kodu. Zwróć uwagę, jak i w którym momencie Hibernate wywołuje zapytania do bazy danych
3. W klasie `CourseEntityDaoExt` uzupełnij metody, które wykorzystają JPQL, zgodnie z tym co zapisane jest w komentarzach do metod:
   a. `findByCity()`,
   b. `findByName()`,
   c. `findByDateRange()`,
   d. `findByCities()`
4. W klasie `StudentEntityDaoExt` uzupełnij metody, które wykorzystają JPQL, zgodnie z tym co zapisane jest w komentarzach do metod:
   a. `findBySeatRow()`,
   b. `findByCityAddress()`,
   c. * `findBySkills()`,
   d. * `findTheMostSkilled()`