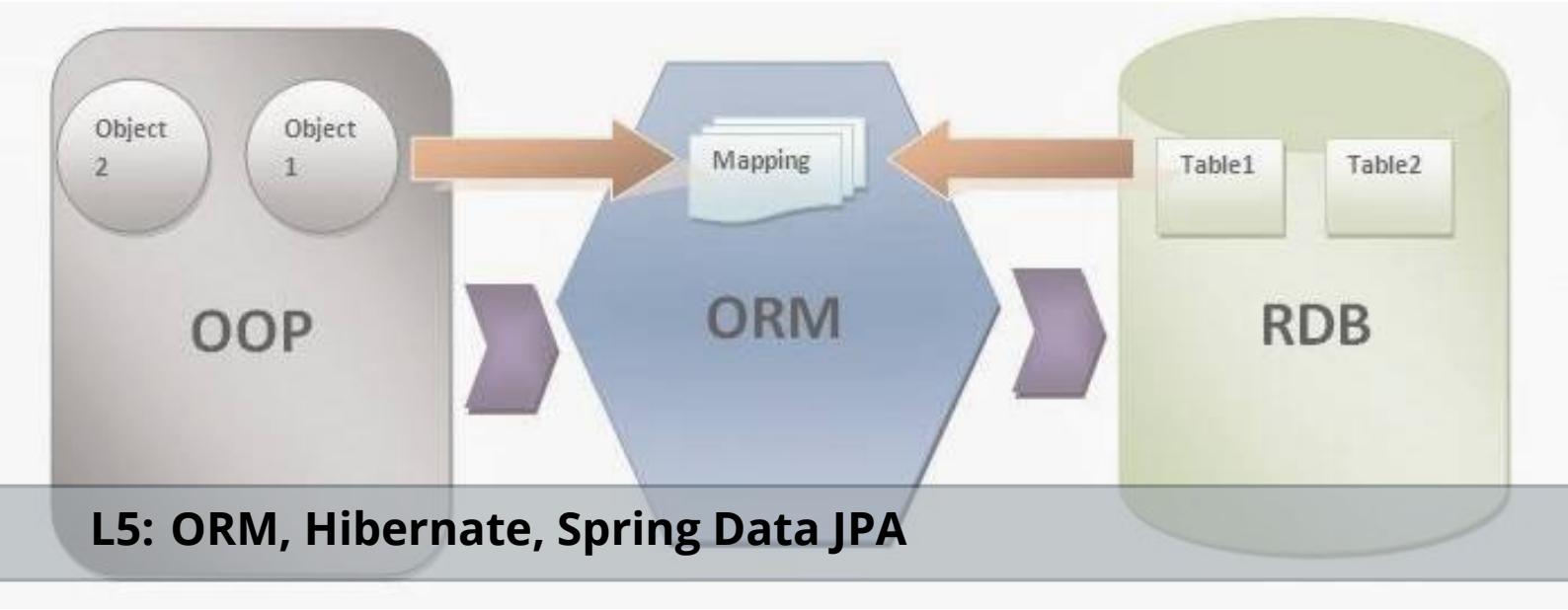

L5: ORM, Hibernate, Spring Data JPA

Adatelérési réteg JPA-val

- A jelen laborfeladatban egy harmadik adatelérési implementációt építünk be a meglévő webalkalmazásunkba, ezúttal egy objektum-relációs leképezési eszközt (Hibernate) alkalmazva. Ezzel együtt a releváns adatelérési műveleteket generáljuk modellek és annotációk alapján (Spring Data JPA).
- A laborfeladatok továbbra is használják az első feladatban előírt projekttematikát (ld. `ubb-idde-lab1-build.pdf`).
- Továbbra is feltétel, hogy ne legyenek statikus kódelemző jelzések, illetve hogy helyesen fusson le a `check` taszk a folyamatos integrációban (ld. korábbi laborfeladatok leírása).
- Az alábbi leírások általánosan érvényesek mindenkire, de mindenki alkalmazza őket azokra az entitásokra/funkcionalitásokra, melyek már léteznek a korábbi laborokról. Ha valahol az általános leírás nem tisztán alkalmazható a meglévő projektre, kérdezzétek a laboránsokat tisztázásért.

Java Persistence API

Java Persistence API

- Vezessük be a Spring Data JPA függőséget a starter csomagján keresztül. Ez hozza magával a JPA-t és Hibernate-et, de az adatbázis-specifikus JDBC drivert *nem*–eszerint építsük fel a függőségi listánkat.
- Konfiguráljuk az adatbázis-elérést Spring által nyújtott property-k megadásával, vagy írjuk fölül a `DataSource` Spring osztály bean definició függvényét, hogy használja a korábban definiált konfigurációs kulcsokat.
  - *Megjegyzés:* Ha már fölülírtuk a `javax.sql.DataSource` bean definicióját pl. HikariCP segítségével, az továbbra is működőképes megoldás, csak tegyük aktívvá a bean definiciót az új profil esetén is.
- Annotáljuk a modellosztályainkat a megfelelő JPA annotációkkal. Vigyázat, hogy a modellek szülőosztályában definiált elsődleges kulcs is kerüljön be a táblákba, illetve automatikus értéket kapjon beszúráskor.
- A megfelelő adatbázis-táblákat a rendszer **hozza létre automatikusan**, s loggolja a generált SQL parancsokat.

Spring Data JPA adatelérési réteg

Spring Data JPA adatelérési réteg

- Készítsünk egy Spring által generált adatelérési réteget: `@Repository`-val annotált DAO interfészeket, melyekhez a Spring generál futási időben implementációt. Válthassunk erre az adatelérési módszerre Spring profil beállításának segítségével.
  - *Megjegyzés:* A korábbi profilok is kell működjenek. Ha az autokonfigurációs mechanizmus miatt nem működnek, ki lehet zárni a Spring Data JPA autokonfigurációját *csak a JDBC-mem profilokban (konfigból)* – ld. `https://www.baeldung.com/spring-data-disable-auto-config`
- Az új DAO interfészek s a régiek közti kompatibilitást megoldhatjuk többféleképpen: Az új Spring Data JPA interfészek örökölhetik a korábbiakat (ebben az esetben megfelelő metódus-átnevezés szükséges a sajátjainkban), vagy használhatunk profilozással leváltható `@Service` osztályokat, melyek delegálnak a régi vagy új interfészek felé.

Kapcsolat jelölése

Kapcsolat

- A korábbi laborban definiált *kapcsolatot* is jelezzük a modell osztályokban, s explicit módon deklaráljuk a két entitás közti kapcsolatban a lekérés mechanizmusát (*fetch type*), illetve a CRUD műveletek továbbítását (*cascading*). Ügyeljünk ezeknek helyes megválasztására.
- Készítsünk egy új REST Controllert, amely kihasználja a két entitás között felállított kapcsolatot. Pl. ha a 2 entitásunk `Menu` és `MenuItem` egy-a-többhöz kapcsolatban vannak:
  - lehessen listázni egy bizonyos menühöz tartozó bejegyzéseket
  - lehessen hozzáadni egy meglévő menühöz egy új bejegyzést
  - lehessen törölni bejegyzést a menün keresztül
- Ezekben az esetekben **ne** használjunk új Spring Data vagy JPQL query-ket, hanem használjuk a leképezett objektumokat.