

JPA, Spring Data JPA

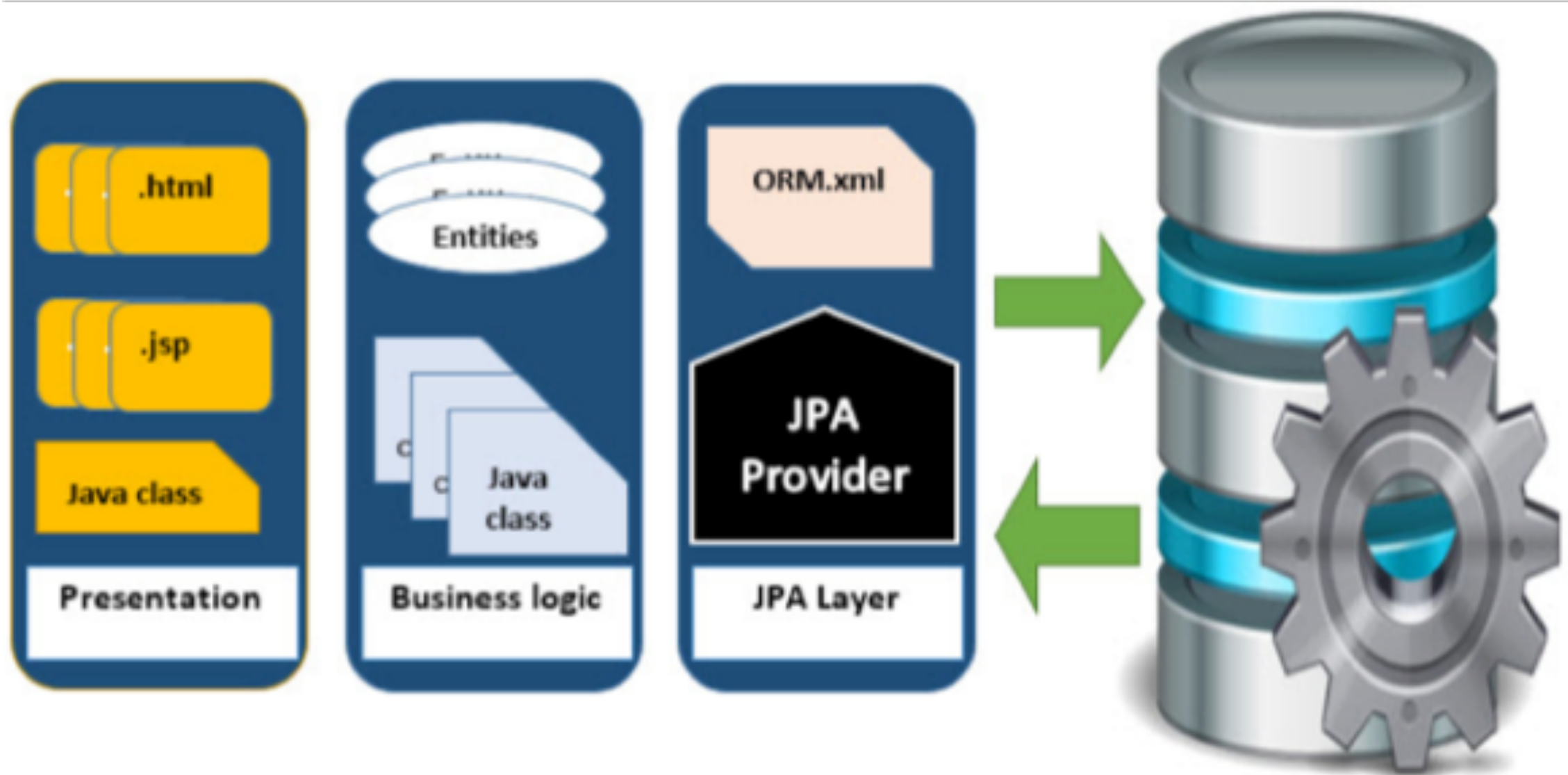
Šta je JPA

- Java Persistence API
- Kolekcija interfejsa/klasa sa metodama za trajno čuvanje (perzistenciju) podataka (u bazama podataka)
- Svaka ozbiljnija aplikacija čuva velike količine podataka, koje je prilikom čuvanja potrebno na odgovarajući način premapirati na (relacioni) model podataka koji se koristi u bazama
- JPA obezbeđuje “most” između objektnog modela podataka u programskom kodu (Java) i relacionog modela u bazi

Zašto koristiti JPA

- Smanjuje količinu koda koji je neophodno pisati kako bi se obezbedilo snimanje podataka u bazu
- Redukuje greške koje mogu nastati pri pisanju koda specifičnog za određenu bazu
- Programer se koncentriše na rad sa objektnim modelom
- Programer sledi pravila JPA provider framework-a koji za njega kreira DDL/SQL komande

Gde se koristi JPA



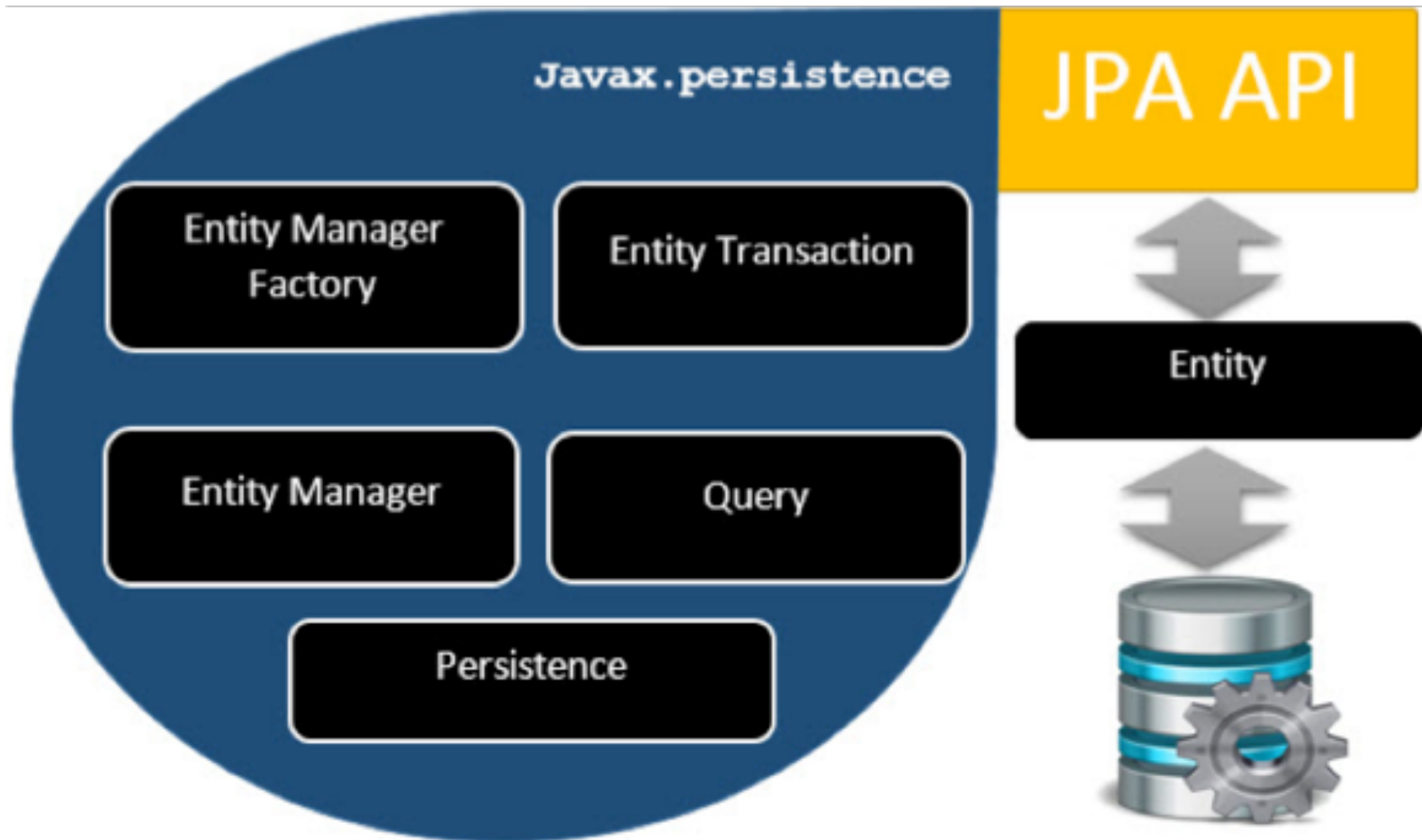
Istorija JPA

- Potiče iz Enterprise Java Beans (EJB) specifikacije
- Ranije verzije EJB su sloj za perzistiranje podataka mešale sa slojem poslovne logike
- Tokom razvoja verzije EJB 3.0 sloj za perzistenciju podataka je izdvojen i postao je JPA v 1.0

JPA Provideri

- JPA je API otvorenog koda (open source), tako da postoji više implementacija
 - više proizvođača - Oracle, RedHat, Eclipse
 - više frameworka - Hibernate (najpopularniji), Eclipselink, Toplink, Spring Data JPA

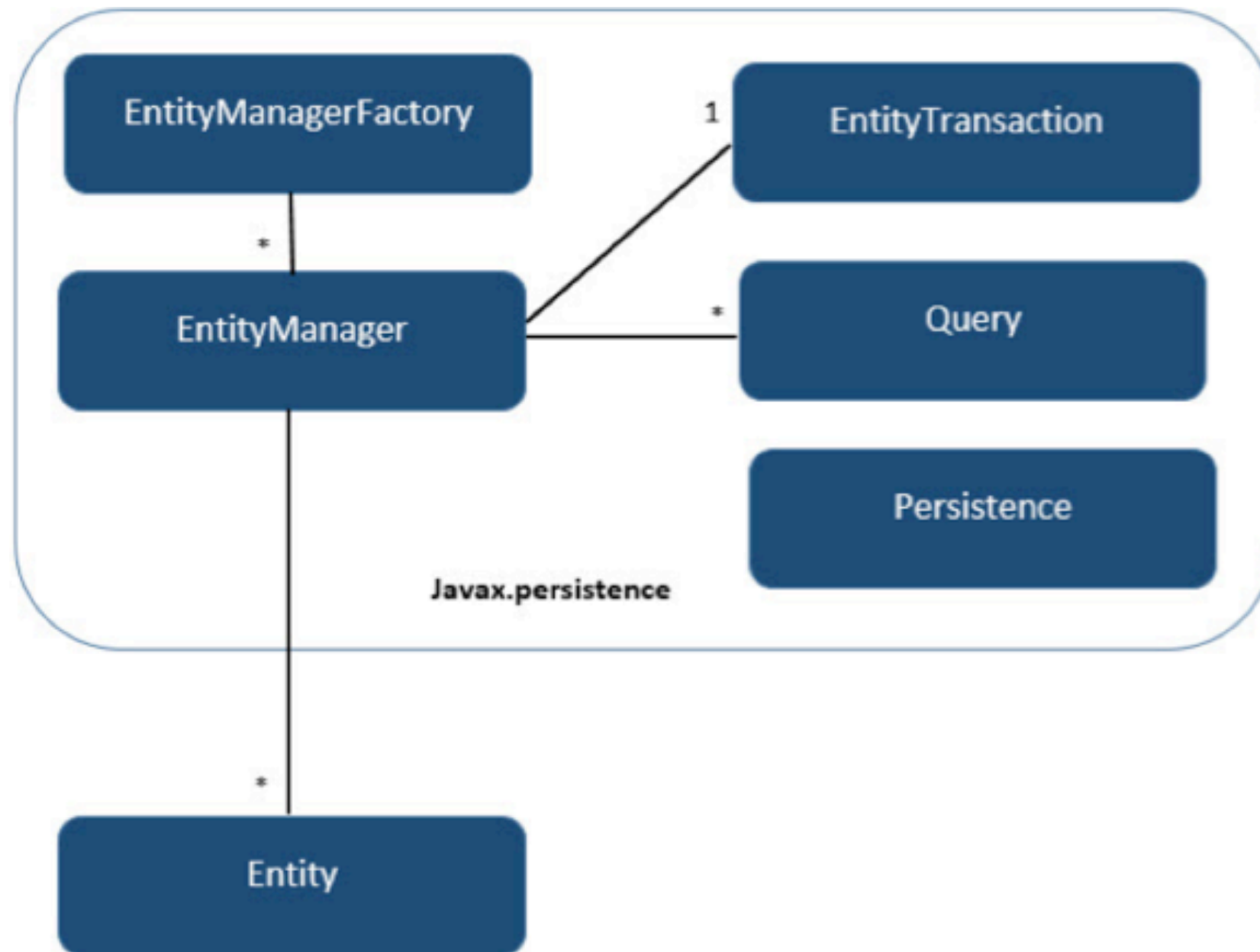
Class level arhitektura



Class level arhitektura

- paket *javax.persistence*
- **EntityManagerFactory** - kreira i upravlja instancama EntityManager-a
- **EntityManager** - interfejs, upravlja operacijama perzistiranja podataka. Instance EntityManager-a služe za kreiranje instanci Query-ja
- **Entity** - objekti koji se perzistiraju - preslikavaju se na zapise u bazi podataka. Entiteti predstavljaju domenske podatke specifične za vaš program
- **EntityTransaction** - Za svaki EntityManager jedan EntityTransaction upravlja izvršavanjem operacija
- **Persistence** - obezbeđuje statičke metode putem kojih se pribavljaju instance EntityManagerFactory-ja
- **Query** - svaki JPA provider implementira Query interfejs na način da se zadovolje specifični zahtevi i da se zadovolje kriterijumi

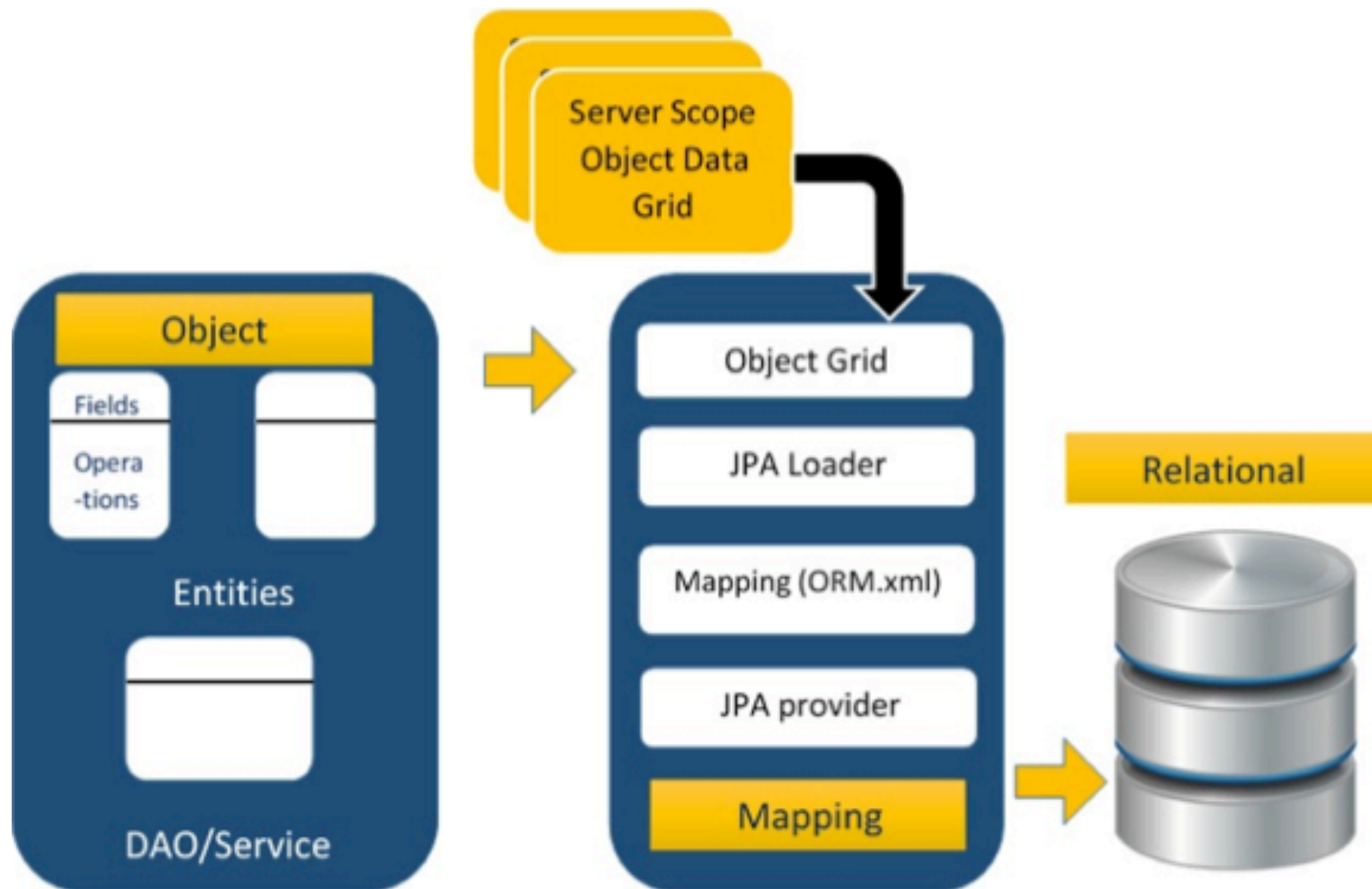
Class level arhitektura



Object Relational Mapping

- Obezbeđuje povezivanje (preslikavanje, mapiranje) podataka iz objektnog modela na podatke u bazi podataka
- Mapiranje mora da uzme u obzir
 - tipove podataka
 - veze između podataka

ORM arhitektura



ORM - faze mapiranja

Podaci prolaze kroz nekoliko faza

1. faza objektnog modela podataka - POJO klase - entiteti, servisne klase koje nad njima obavljaju određenu poslovnu logiku, menjajući stanje objekata
2. faza mapiranja
 - JPA Provider - specifična implementacija javax.persistence api-ja koju obezbeđuje neki proizvođač
 - pravila mapiranja - opisana mapping fajlovima (mapping.xml), ili u novije vreme anotacijama
 - JPA Loader - funkcioniše kao cache memorija, sa kopijom dela baze podataka
 - Object Grid - lokacija za privremeno čuvanje podataka. Svi upiti prema bazi se prvo reflektuju nad ovim podacima
3. faza relacionog modela podataka - fizičko smeštanje podataka u tabele baze

JPA anotacije

- Mapiranje pomoću zasebnih xml dokumenata je zahtevalo da se za jedan entitet paralelno održavaju dva dokumenta, jedan sa Java klasom i jedan sa pravilima mapiranja
- Ako je dolazilo do izmene Java klase bilo je neophodno ručno održavati ove fajlove kako bi mapiranje bilo ažurno
- Anotacije rešavaju ovaj problem jer se kompletno mapiranje opisuje pomoćnim anotacijama
- Sve anotacije su definisane u javax.persistence paketu

JPA anotacije

- **@Entity** - deklariše klasu kao entitet - biće preslikana na tabelu u relacionom modelu baze podataka
- **@Table** - deklariše naziv tabele u bazi koja će biti korišćena za prezistiranje entiteta
- **@Basic** - koristi se da eksplicitno označi property koji treba da se perzistira, namenjen je za jednostavne tipove. Ovo je i defaultno ponašanje - property koji se ne anotira nikako drugačije biće tretiran kao basic
- **@Embedded** - specificira da je dati property instanca neke druge klase

JPA anotacije

- **@Id** - označava property koji se koristi kao identifikator (primarni ključ tabele u bazi)
- **@GeneratedValue** - specificira kako identifikator može biti inicijalizovan (automatski, ručno, pomoću sekvence...)
- **@Transient** - specificira da se dati property ne perzistira - vrednosti se nikada ne snimaju u bazu - pogodno za neke propertyje u kojima se čuvaju privremene ili kalkulisane vrednosti
- **@Column** - specificira da se dati property mapira na kolonu određenog naziva

JPA anotacije

- **@SequenceGenerator** - koristi se u kombinaciji sa **@GeneratedValue** da specificira upotrebu generatora sekvenci za identifikatore
- **@TableGenerator** - koristi se u kombinaciji sa **@GeneratedValue** -da specificira upotrebu tabele za identifikatore
- **@AccessType** - specificira način pristupa određenoj vrednosti
 - **AccessType.FIELD** - pristupa se direktno preko atributa (podrazumevano)
 - **AccessType.PROPERTY** - pristupa se preko getter funkcije - ovo omogućava da se ugradi neka dodatna logika

JPA anotacije

- **@JoinColumn** - koristi se da se specificira one-to-many ili many-to-one veza između entiteta - omogućava da se navede koja kolona predstavlja strani ključ
- **@UniqueConstraint** - omogućava da se postave pravila za jedinstvenost određenih atributa (u koloni u tabeli u bazi na koju se mapira ovaj atribut se u tom slučaju se ne mogu naći više puta iste vrednosti)
- **@ColumnResult** - referencira naziv kolone u result setu SQL upita

JPA anotacije

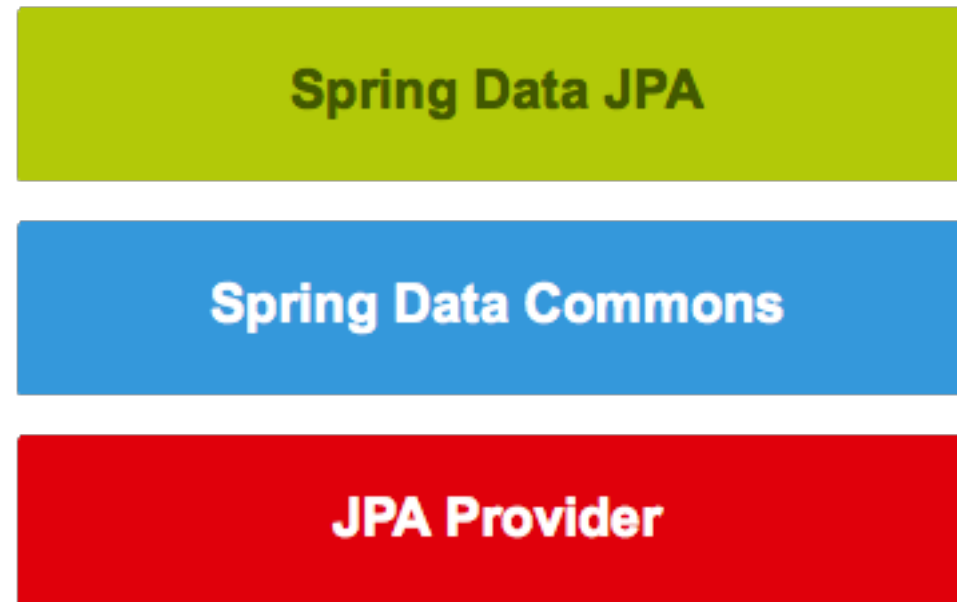
- **@OneToOne** - definiše relaciju 1-1 između entiteta (spojenih tabela)
- **@OneToMany** - definiše relaciju 1-N između entiteta (spojenih tabela)
- **@ManyToOne** - definiše relaciju N-1 između entiteta (spojenih tabela)
- **@ManyToMany** - definiše relaciju N-N između entiteta (spojenih tabela - ovde se mora navesti specifikacija strukture vezne tabele)
- **@NamedQuery** - Specificira query pomoću statičkog naziva
- **@NamedQueries** - specificira listu imenovanih upita

Spring Data JPA

- Cilj je pojednostaviti primenu JPA time što ćemo izbeći pisanje prostih operacija za svaki entitet.
- Kada se radi sa JPA za svaki entitet:
 - Kreira se apstraktni Repository za svaki entitet koji obezbeđuje CRUD operacije
 - Kreira se konkretna implementacija za apstraktne repozitorijume kako bi se implementirale specifične metode

Spring Data JPA

- Nije JPA Provider, već biblioteka koja dodaje dodatni sloj apstrakcije nad JPA providerom, sada aplikacioni sloj za perzistenciju ima tri sloja



Spring Data JPA

- Spring Data JPA - obezbeđuje podršku za kreiranje JPA repozitorijuma nasleđivanjem Spring Data Repository interfejsa
- Spring Data Commons - obezbeđuje neophodne klase za podršku
- JPA Provider implementira sam JPA api

Spring Data JPA

- Spring data commons:
- *Repository<T, ID extends Serializable>* interface je marker interface:
 1. definiše tip upravljanog entiteta, kao i tip vrednosti koja se koristi za ID.
 2. pomaže Spring kontejneru da tokom faze classpath skeniranja pronađe konkretne primerke repository interface-a.
- *CrudRepository<T, ID extends Serializable>* interface obezbeđuje CRUD operacije za upravljani entitet.
- *PagingAndSortingRepository<T, ID extends Serializable>* interface deklariše metode koje se koriste za sortiranje i paginaciju podataka preuzetih iz baze.
- *JpaRepository<T, ID extends Serializable>* interface je verzija interfjesa specifična za određenu implementaciju JPA.

Spring Data JPA

- Spring data JPA:
- *QueryDslPredicateExecutor<T>* interface nije u pravom smislu “repository interface”. Deklariše metode koje se koriste da se korišćenjem QueryDsl Predicate preuzmu entitei iz baze.
- *JpaSpecificationExecutor<T>* interface nije u pravom smislu “repository interface”. Deklariše metode koje se koriste da se eniteti preuzmu iz baze korišćenjem *Specification<T>* objekata koji koriste JPA *criteria API*.

Spring Data JPA

- Primer:

```
import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demo.entities.User;
```

@Repository

```
public interface UserRepository extends
JpaRepository<User, Long> {
    public User findByUsername(String username);
    public User findByEmail(String email);
}
```


@Repository

- @Repository je Spring anotacija koja označava da klasa reprezentuje repozitorijum podataka.
- Repozitorijum podataka enkapsulira funkcionalnosti čuvanja, čitanja i pretraživanja podataka emulirajući rad nad kolekcijom objekata.

@Repository

- U prethodnom primeru klasa **UserRepository** predstavlja repozitorijum podataka za objekte tipa **User**, čiji identifikator je vrednost **Long** tipa
- Ovaj repozitorijum omogućava korisniku pretraživanje po username-u i email-u

@Repository

```
public interface UserRepository extends  
JpaRepository<User,Long> {  
    public User findByUsername(String username);  
    public User findByEmail(String email);  
}
```

@Repository

- Ukoliko se drži konvencije imenovanja funkcija dovoljno je samo deklarirati funkcije za pretraživanje, implementacija će biti automatska
- ***List<Person> findByLastname(String lastname);***
podrazumeva da u Entity-ju **Person** postoji property **lastname**, i obaviće pretragu poredeći odgovarajuću kolonu (koja je u Entity-ju mapirana na lastname)

@Repository

- Još primera

@Repository

```
public interface PersonRepository extends Repository<Person, Long> {
```

```
    List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String  
lastname);
```

```
    // Enables the distinct flag for the query  
    List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String  
firstname);
```

```
    List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String  
firstname);
```

```
    // Enabling ignoring case for an individual property  
    List<Person> findByLastnameIgnoreCase(String lastname);  
    // Enabling ignoring case for all suitable properties  
    List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String  
firstname);
```

```
    // Enabling static ORDER BY for a query  
    List<Person> findByLastnameOrderByFirstnameAsc(String lastname);  
    List<Person> findByLastnameOrderByFirstnameDesc(String lastname);  
}
```