Šta je ORM?

- Object Relational Mapping
- Baratamo klasama i objektima
  - tzv. POPO: Plain Old Php Object
- To se, u pozadini, *prevede* na operacije sa bazom
  - Dodavanje redova
  - Stvaranje veza
  - Dovlačenje podataka iz DB -- prosti i složeni upiti
  - modifikovanje baze...
- obično, 1 tabela -> 1 klasa (entitet)
- Entiteti, domenske klase = klase koje ORM perzistira DB

Zašto ORM?

- Principi razvoja:
  - **DRY**: *Don't Repeat Yourself*
    - bez ORM: kod sadrži dosta ponavljanja (otvaranje konekcije ka DB, formiranje upita, *escaping* vrednosti koje se ugrađuju u upit, dohvatanje rezultata...)
  - *Don't Reinvent The Wheel*
    - Može se pasti u iskušenje da se piše kod koji je "čist", bez ponavljanja i možda Objektno Orijentisan
    - To već postoji, testirano je i korišćeno. Vreme upotrebiti za razvoj aplikacije.

Zašto ORM?

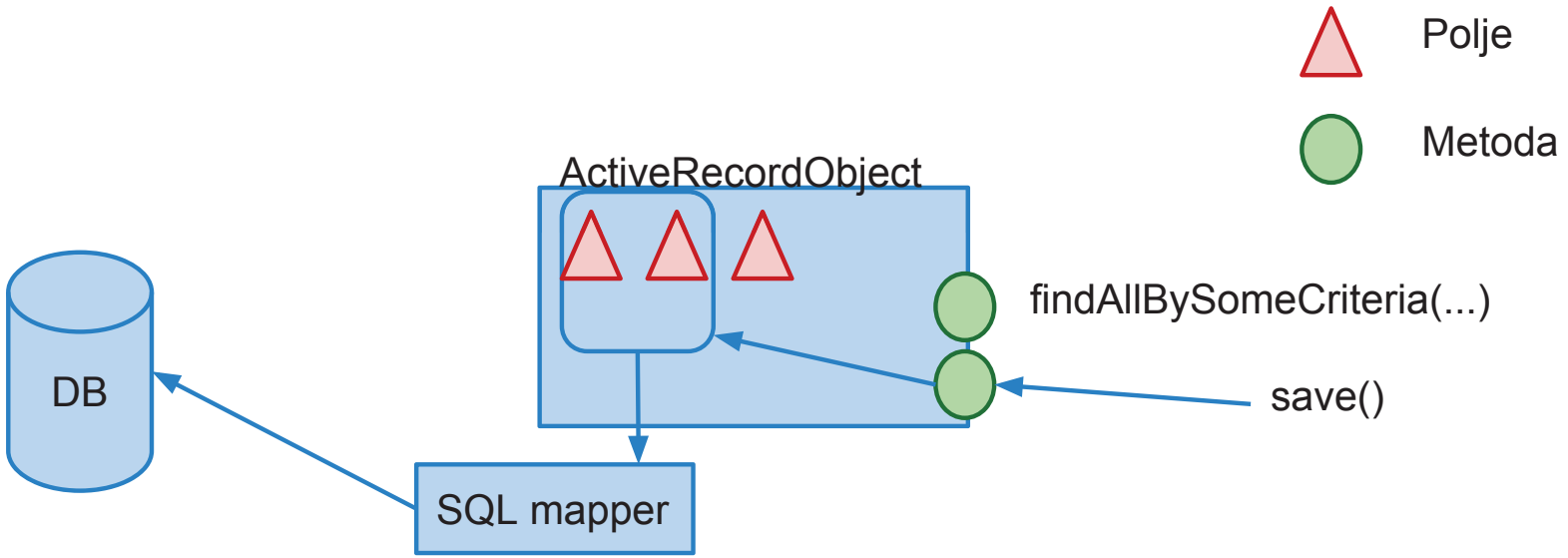
- Piše se na visokom nivou apstrakcije
- Radi se sa objektima
- Moguće lakše testiranje koda business logike (nezavisno od baze podataka)
- Moguća laka promena baze (MySQL na PSQL npr), ORM prelaz sređuje sam!
- Nema SQL stringova rasutih po čitavom kodu
  - jedna od loših praksi, široko raširenih u PHP programiranju

Zašto ORM?

- Zaštita od sql injection-a - ORM sredi
  - `umesto mysqli::real_escape_string()` ili `PDO::quote()`, koji se mora pozivati za svaki parametar koji želimo da ugradimo u upit
    - Da nam neko ne bi poturio `drop database` i sve "sredio"
- Zato što je to standard u industriji
  - Svaka tehnologija ima ORM framewoke: JPA, Hibernate, RoR ActiveRecord, Entity Framework...
  - I za jednu tehnologiju postoji dosta različitih rešenja
  - Iole kompleksniji sistemi mahom koriste *neki* ORM, retko se radi bez
- Pitanje kompleksnosti i zahtevnosti (memorija, brzina...) -nije više problem

ORM varijante: Active Record

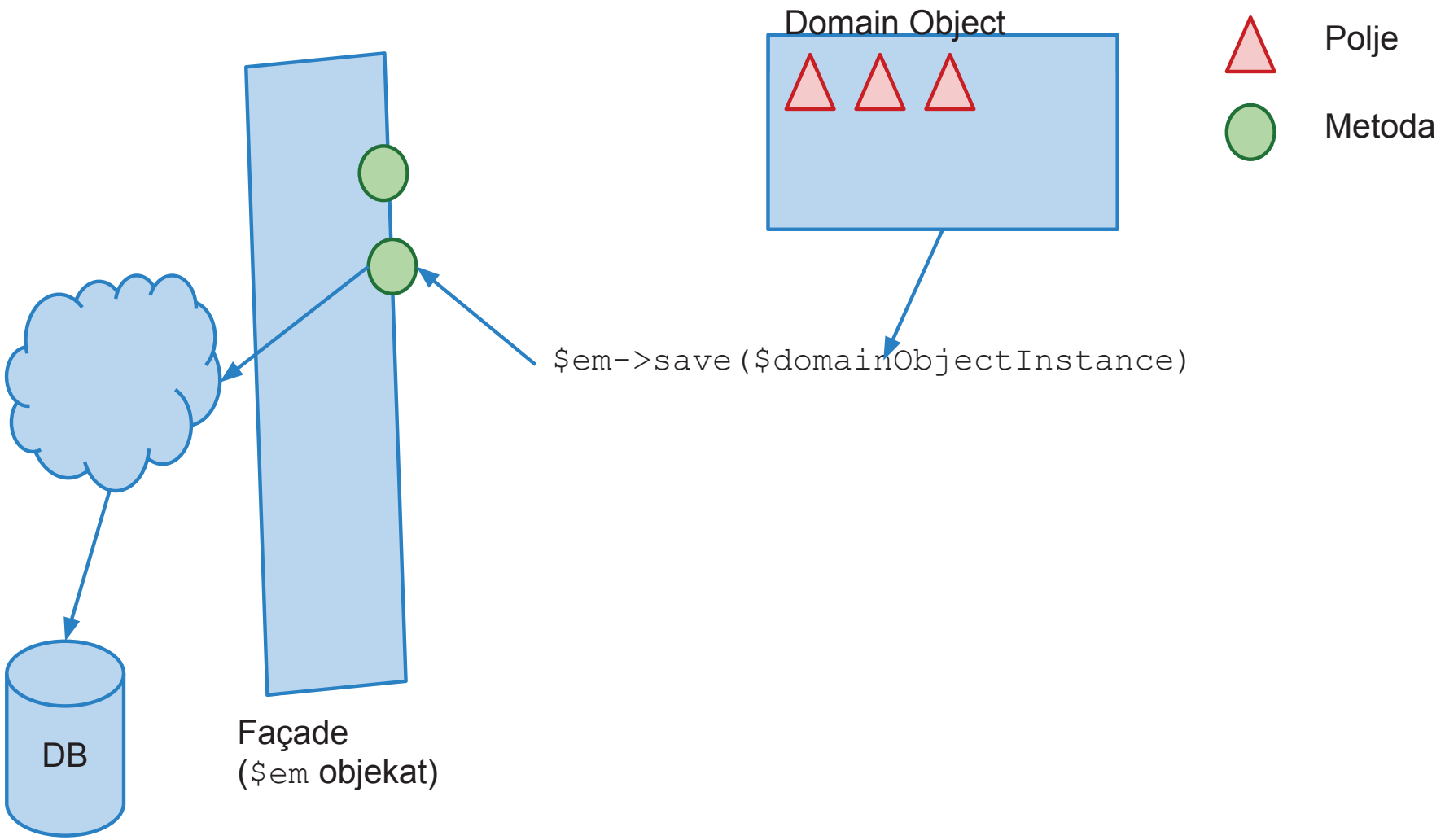
- *Aktivni zapis*, ako moramo prevoditi
- Objekat koji sam pruža metode,
  čijim se pozivanjem izmene nad njim perzistiraju
  (šalju u bazu, i time čine trajnim): save, delete...
- Nudi metode koje rade perzistiranje određenih polja
  - nije POPO, svestan je da služi za perzistiranje



ORM varijante: Data Mapper

- Takođe jako popularan
- Može biti osnova za activerecord
  (da stoji u pozadini)
- Skup klasa, dat jednom fasadom,
  koja, na osnovu objekata -entiteta koji se
  predaju, izvodi operacije nad bazom
- Domenski objekti su POPO - samo objekti
  koji čuvaju podatke; ne znaju za postojanje
  baze niti da su perzistirani
  (*database agnostic*)

Entity Manager



ORM varijante - pristup

- Pišemo domenske klase, generišemo bazu
  - U domenskim klasama, kroz anotacije, definišemo kako želimo da izgleda baza, alat analizira klase, generiše DDL
- Pravimo bazu, generišemo klase
  - alat posmatra bazu, i odatle generiše odgovarajuće klase
- Pišemo XML/YAML konfiguracione fajlove, generišemo i bazu i klase
  - kroz konfiguracioni fajl specificiramo izgled baze, alat generiše DDL i generiše domenske klase
- Kako izgleda baza:
  - šta želimo da imamo u entitetima (nazivi polja u klasama kolona u tabelama, tabela, tipovi...)
  - kako su entiteti međusobno povezani -- asocijacije(firma referencira svoje zaposlene...)

Primer: Doctrine

- www.doctrine-project.org
- Data mapper implementacija
- Podešavanje
  - preko *composer* alata (koji se instalira zasebno)
  - Ili skinite sa sajta. Za tutorijal, sve je spakovano
- Konfiguracija: kroz fajlove
  - libraries/Doctrine.php
    - specificira parametre konekcije, kao i koji način za mapiranje se koristi (koriste se anotacije i parametri iz config/databases.php)
  -

Alat

- Putanja do alata, koja se koristi u konzoli
  - `application/doctrine.php`
- Konzolu pokrenuti u onom direktorijumu u kom je **`doctrine.php`**!

Praktično (Demo)

1. Kreiramo bazu `doctrine2`
2. Podesimo `bootstrap.php`, tako da imamo valjane parametre konekcije
3. Definišemo klasu `User i` par polja (`id, name`)
4. Malo anotacija (v. naredne slajdove i primere)

   Komentari sa malo pravila u pisanju - govore doctrine kako generisati bazu
5. Naredba koja generiše tabele koje odgovaraju klasama

```
php application\doctrine orm:schema-tool:create --dump-sql
```

6. Naredba

```
php application\doctrine orm:schema-tool:create
```

Praktično (Demo)

- Neophodno je dodati putanju do php-a u PATH
- Linux: već je tu :)
- Windows -ako se javi greška

```
'php.exe' is not recognized as an internal or external command,
operable program or batch file
```

Pratiti http://support.microsoft.com/kb/310519, dodajte user varijablu, ime: `PATH`, vrednost: putanja do `php.exe`

```
C:\wamp\bin\php\php5.x.y\bin\php.exe
```

ako koristite WAMP, gde je X.Y podverzija PHP-a

- U **novoj** konzoli probati komandu sa prethodnog slajda
  (konzola ne vidi izmene u environent promenljivim, nakon startovanja, zato nova!)

Anotacija

Parametrizovana anotacija

Indicira da je potrebno napraviti tabelu koja čuva objekte ove klase. Bez ovoga, neće biti tabele

```
/**
 * @Entity @Table(name="persons")
 */
class Person
{
    /** @Id @Generated
     *  @Column(type="integer")
     */
    private $id;
    /** @Column() */
    private $username;
}
```

Želimo da se ID generiše automatski (doctrine koristi autoincrement tada) -postoji više strategija generisanja ID-a

Polje koje čuva primarni ključ (ID)

Ekspliciitno odredimo tip -integer

Samo specificira da se ovo polje mapira kao kolona. Ako se tip ne navede, podrazumeva se da će biti **string**

Anotacije

- Kroz dokumentacione komentare (phpDoc)
  - Dokumentacioni komentar započinje sa 2 zvezdice
    - Pomaže IDE-u da prikaže namenu funkcije, očekivane tipove parametara, povratne vrednosti i namenu funkcije...
  - Anotacije - unutar phpDoc, i imaju prefiks @
- To su metapodaci: podaci o podacima
- Doctrine alalizira tekst klasa, izvlači anotacije, i koristi ih za pravljenje DDL
- Sve što želimo da "poručimo" Doctrine, činimo preko anotacija
  - ali postoje i drugi načini: XML, YAML

Anotacije

- Anotacije polja:
  - `@Id`
  - `@GeneratedValue`
  - `@Column()`
    - Da bi polje koje se anotira dobilo kolonu u tabeli koja se generiše
    - mogući atributi (neki od): `type`, `name`...
- Da bi se sama klasa tretirala kao entitet, i dobila tabelu u bazi, anotirati je sa `@Entity`
- Anotacije su dokumentovane - pogledati dokumentaciju:

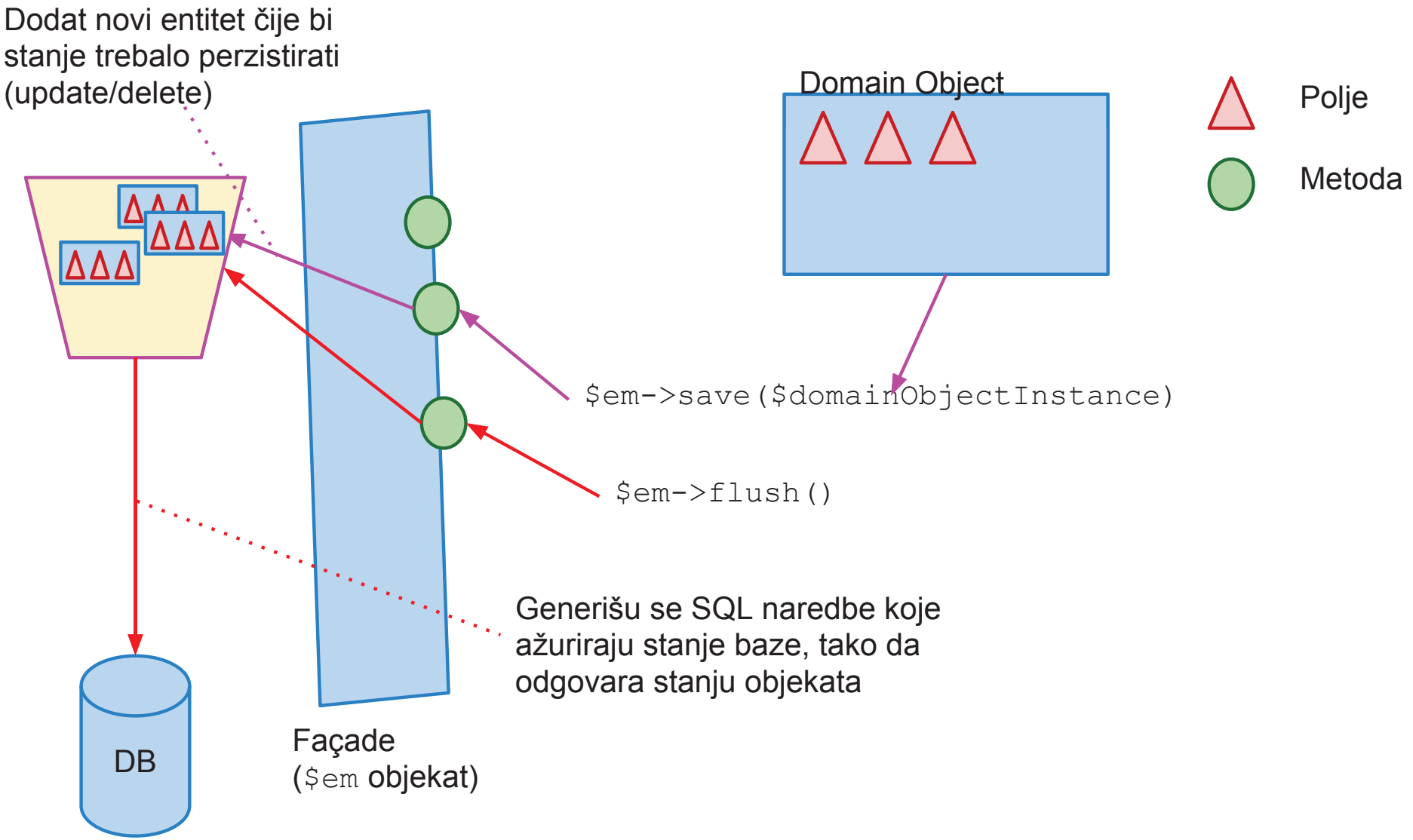
http://docs.doctrine-project.org/en/2.0.x/reference/annotations-reference.html

Entity manager i rad sa objektima

Entity manager

- Radi transformaciju entiteta u zapise u bazi,
  i obratno
- Izvršava upite i narede koje kreiraju, ažuriraju i uklanjaju zapise iz baze, tako da se odraze promene načinjene nad objektima - instancama klasa modela
- Funkcioniše tako što sve insert, update i delete naredbe koje mora da izvši *skuplja*,
  t.j. ne izvši ih istog momenta, jednu po jednu.
- Izvši ih paketski, sve "skupljene" u okviru jedne transakcije
  - Ako se desi problem (npr. referencijalni integritet) **transakcija se rollbackuje**,
    i ništa iz tog paketa ne ostane urađeno

Entity manager i UnitOfWork



Entity manager

- Pogledati libraries/Doctrine.php, u kome se vrši dohvatanje EM
  - Kreira se za konekciju koju imamo
- U kontroleru:
  - Doctrine se učitava kao biblioteka: $this->load->library(‘doctrine’)
  - Time dobijamo property doctrine, preko kog možemo dohvatiti EntityManager-a sa kojim se radi:

```
$em = $this->doctrine->em
```

- Kreiramo novi objekat (markiran sa `@Entity`)
- Pozovemo `$em->persist()` - EM *sačuva* informaciju da treba da kreira novi objekat
- Ako ne pozovemo `$em->flush()`, neće se desiti izmene na DB
- Pozivom `flush`, izazivamo izvršavanje svih update/create/delete akcija nad bazom, koje su bi trebalo da sačuvaju tekuće stanje objekata
  (to su one "sačuvane informacije")

Entity manager

Kreiramo objekte -novi su, ne potiču iz baze
Moramo EM-u staviti do znanja da ih mora sačuvati

Dostavljamo novokreirani objekat -EM zna da bi trebalo da izvrši insert

Biće sačuvani samo `$nick` i `$mick`, jer su dostavljeni EM-u pre `flush`;

`$jo` će biti sačuvan u DB, ako se u kodu koji dolazi, javi poziv `flush` metode

```
// some persons
$nick = new Person();
$nick->setName('Nikola Tesla');


$mick = new Person();
$mick->setName('Mihajlo Pupin');


$jo = new Person();
$jo->setName('Jovan Cvijic');


$entityManager->persist($nick);
$entityManager->persist($mick);
$entityManager->flush();
$entityManager->persist($jo);
```

Dohvatanje i izmena postojećeg objekta

- Dohvatanje entiteta `Person`, sa ID 1:

```
$p1 = $entityManager->find("Person", 1);
echo $p1->getName();
$p1->setName("Novo ime");
$entityManager->flush();
```

- Ime će biti promenjeno (update će se desiti), iako nije pozvan `persist()`
- Zašto?

Entity manager

- Zato što...
  - EM je *dovukao* `p1` iz baze, i vodi evidenciju o tome
    - Postoji skup svih objekata koji su dovučeni iz DB
    - Otud i naziv - *manage-uje* entitete
  - Kada se radi `flush`,
    proveravaju se svi objekti koji su doneti iz baze
  - Ako je neki izmenjen (*dirty*),
    EM samostalno generiše update naredbu
- Koristi se *identity map* pattern
- Kada zahtevamo entitete sa istim ID-em na više različitih mesta u kodu, dobijamo reference ka istom objektu!

Asocijacije

Referisanje - asocijacije?

- Moguće - referišemo jedan objekat iz drugog,
  a u DB postoji strani ključ koji to oslikava
  - radimo samo sa objektima i kolekcijama objekata
- Tipovi veza
  - 1-1
  - 1-*
  - *-1
  - *-*
- Vlasnik veze (owner)
- Strane u asocijaciji
  - Mapped by
  - Inverted by

Tipovi veza

- **Unidirekcione**: samo u jednoj klasi postoji referenca ka objektu druge
- **Bidirekcione**: u obe klase postoje reference ka drugoj strani
- **Multiplikativnost** - sa koliko objekata na suprotnom kraju je posmatrani entitet u vezi

Veze - primer

- Ima više `Book` objekata kojima je on(a) glavni primarni autor;
  primarnih autora može biti 1, pa je, sa strane `Person` to veza tipa `1-*`, t.j. `OneToMany`.
- Sa strane `Book` objekta, on ima jedan `Person` objekat za glavnog autora,
  dok takvih `Book` objekata može biti više,
  pa se radi o `*-1`, t.j. `ManyToOne` vezi

Kako se definišu asocijacije?

- Putem anotacija
- `@ManyToOne(targetEntity="User", inversedBy="...")`
- `@OneToMany(targetEntity="Bug", mappedBy="...")`
- `@ManyToMany(targetEntity="Book", inversedBy="..")`
- Na primeru je pokazano kako se određuje tip asocijacije
- Atributi `mappedBy` i `inversedBy` u anotacijama moraju sadržati naziv polja unutar klase `targetEntity`, koje sadrži referencu ka objektu posmatranog entiteta

Primer: 1-* i *-*

```
class Person
{
...
/**
 * @OneToMany(targetEntity="Book",
 *          mappedBy="primaryAuthor")
 */
protected $booksAuthored;

/**
 * @ManyToMany(targetEntity="Book",
 *             mappedBy="coauthors")
 * @var Book[]
 */
protected $booksCoAuthored = null;
...
}
```

```
class Book
{
...
/**
 * @ManyToOne(targetEntity="Person",
 *          inversedBy="booksAuthored")
 */
 protected $primaryAuthor;

/**
 *
 * @ManyToMany(targetEntity="Person",
 *        inversedBy="booksCoAuthored")
 */
 protected $coauthors;
...
}
```

Kako se programski rukuje asocijacijama

- Uz malo više angažmana
- Zavisi od ORM framework-a
  - Neki sve srede iza kulisa (Hibernate, t.j. JPA)
  - Kod nekih (doctrine), dosta toga mora ručno
  - Doctrine: pri stvaranju asocijacije, ručno se moraju obe strane "srediti" kod bidirekcione veze, t.j. samo ona koja referiše, kod unidirekcione - potrebno je dodati objekte u odgovarajuće kolekcije (nizove)
- Obrati pažnju: ***lazy loading***
  - Objekti koji su u asocijaciji, dovlače se po potrebi (pri prvom pristupu).
  - Proxy pattern (*Posrednik*). Ponekad loše: N+1 fetch

Owner asocijacije

- Vlasnik asocijacije
- Doctrine (i mnogi):
  čuvanje izmena owner-a dovodi do perzistiranja u izmenama asocijacije (dodate nove veze, uklonjene postojeće...)
  - Ako se **samo** inverzna (ne-owner) strana izmeni, asocijacija **neće** biti prezistirnana
- Nasuprot *owner*-u stoji inverzna strana (*inverse*)

Owner asocijacije

- Kod owner-a, atributom ***inversedBy*** se specificira polje preko kog inverzna strana referiše svog owner-a (ili više njih, zavisno od kardinalnosti)
- U slučaju 1-*, *-1 veza
  - owner je uvek * strana, jer je kod nje FK (pogledati izgled baze)
- Kod *-* veza: obe strane mogu biti owner-i
  - Tada se logički bira koja strana je odgovorna za rukovanje asocijacijom - da li je knjiga odgovorna da čuva asocijacije sa osobama koje su koautori ili je osoba odgovorna za to?
  - Svejedno,
    jer će se asocijacija perzistirati kad god se odabrani owner izmeni;
    logički gledano - nije svejedno.

Veze kod Doctrine

- Neophodno ručno postaviti sve reference objektima, a doctrine će to smestiti kako treba u bazu
- Primer: book i author. Owner je Book, gde je FK

```
class Book{...
 public function setPrimaryAuthor($primaryAuthor){
    $this->primaryAuthor = $primaryAuthor;
    $primaryAuthor->authoredBook($this);
 }...
}
//-------------------------------------
class Person{...
 public function authoredBook($book){
    $this->booksAuthored[] = $book;
 }...}
```

Postaviti primarnog autora u Book
-jedna referenca sređena

Obavestiti autora da je dobio još jednu knjigu čiji je autor, kako bi i kod njega reference bile sređene

Autor biva obavešten da je dobio još jednu knjigu, i dodaje je u kolekciju (niz!) svojih knjiga

Paziti da se ne upadne u rekurziju (ne sme se pozvati `$book->setPrimaryAuthor($this)` odavde!

Veze kod Doctrine

- Moguće je ***podesiti*** kaskadno ponašanje prilikom rada sa poljima koja učestvuju u formiranju asocijacije
  - podrazumevano: isključeno je
- Tada nije neophodno “ručno” podešavati sadržaj kolekcija
- Ipak se entiteti dovlače u memoriju, pa može doći do veće upotrebe resursa

```
@OneToMany(targetEntity="Book", mappedBy="primaryAuthor",
cascade={"persist", "remove"})
protected booksAuthored;
```

Upiti kod ORM

Upiti - DQL

- ORM obično "donosi" svoju varijantu upitnog jezika
  - HQL, JPQL...
- Doctrine:

```
$dql = "SELECT b, pa FROM Book b join b.primaryAuthor pa";


$query = $entityManager->createQuery($dql);
$query->setMaxResults(30);
$books = $query->getResult();
```

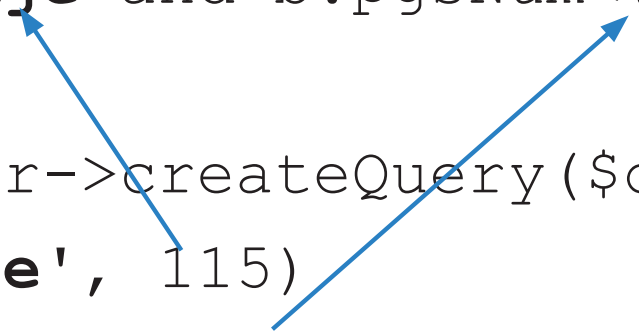
- U upitu figurišu domenske klase (`Book` i sl), i koriste se njihova polja (`primaryAuthor`)
- Moguće je ići “dublje” u pristupu poljima
- (`where pa.name...`)

Upiti - DQL

- imenovani parametri (named parameters)

```
$dql = "SELECT b, pa".
"FROM Book b join b.primaryAuthor pa ".
"where b.pgsNum>:donje and b.pgsNum<:gornje";

$res = $entityManager->createQuery($dql)
->setParameter('donje', 115)
->setParameter('gornje', 155)
->getResult();
```



- Postoje i tzv. pozicioni parametri

```
"...where b.pgsNum>?1 and b.pgsNum<?2";
```

Join

```
$dql = "SELECT b FROM Book b ".
"join b.primaryAuthor pa ".
"left join b.coauthors";
```

sve knjige su vraćene, i one bez koautora

```
$dql = "SELECT b FROM Book b ".
"join b.primaryAuthor pa ".
"join b.coauthors";
```

Vraćene samo one kod kojih je **join** uradio nešto (samo sa koautorima)

Join

- Jednostavan način da se postave upiti koji vraćaju jednu stranu koja *može* učestvovati u asocijaciji; radi se o prirodnom spajanju

```
Book b join b.primaryAuthor pa
```

- Dohvata sve knjige koje imaju postavljenog primarnog autora (sve su takve, doduše)
  - Ako bi postojale knjige bez primarnih autora, ne bi bile vraćene
- ...

Upiti - Query Builder

- `$em->createQueryBuilder()`
- Objekat, kome se "dodaju" delovi upita (metoda `add`) - *Builder* pattern (Graditelj)
- Pruža način da se SQL konstrukcije dodaju pozivom odgovarajućih metoda
- Samo konstruiše upit, programski; izvršavanje se pokreće eksplicitno
- Moguće formirati kompleksne upite, sa opcionim delovima, na osvnovu onoga što korisnik unosi na formi, koristeći if-ove i pozive metoda (uslovno)
  - sa DQL bi se konkatenacija stringova obavljala uslovno... loše i naporno; ovde se uslovno dodaju komponente upitu

Upiti - Query builder

```
$qb = $em->createQueryBuilder();


$qb->select('u')
   ->from('User', 'u')
   ->where('u.name like :name');
// izlged upita (za sada): select u from User u where u.name like :name
if(checkAge){
// opciono dodavanje and where; jednostavnije i preglednije nego
konkatenirati string “and where…”
    $qb->andWhere('u.age> :age');
    $qb->setParameter('age', $ageLimit);
    // upit: select u from User u where u.name like :name and u.age>:age
}
$qb->orderBy('u.name ASC');
$qb->setParameter('name', '%'.$nameSearch.'%');
$result = $qb->getQuery()->getResult();
```

Upiti - Query builder

- Delovi upita kao što su poređenja i slično mogu se ostvariti takođe programski, koristeći expr:
- U potpunosti se izbegava pisanje DQL/SQL koda

```
$qb->select(array('u'))
   ->from('User', 'u')
   ->where($qb->expr()->orx(
       $qb->expr()->eq('u.id', '?1'),
       $qb->expr()->like('u.nickname', '?2')
   ))
```

```
$qb->select(array('u'))
   ->from('User', 'u')
   ->where($qb->expr()->orx(
       $qb->expr()->eq('u.id', '?1'),
       $qb->expr()->like('u.nickname', '?2')
   ))->orderBy('u.name', 'ASC'));
```

Korisni linkovi

- http://docs.doctrine-project.org/en/latest/ - Doctrine
- http://en.wikipedia.org/wiki/Object-relational_mapping - Opšti članak o ORM
- http://www.martinfowler.com/eaaCatalog/activeRecord.html - Opis ActiveRecord pattern-a (Enterprize Architecture Patterns, M. Fowler)
- http://en.wikipedia.org/wiki/Active_record_pattern - Opšti članak ActiveRecord pattern-a
- http://redbeanphp.com/
  Alternativni, lagani ORM framework za php
- http://blog.codinghorror.com/object-relational-mapping-is-the-vietnam-of-computer-science/
  Kritički osvrt na ORM
- Codeigniter podrška ORM-u:
  - http://ellislab.com/codeigniter/user-guide/database/active_record.html - Active record klasa kod CI
  - http://ellislab.com/codeigniter/user-guide/general/models.html Modeli kod CI