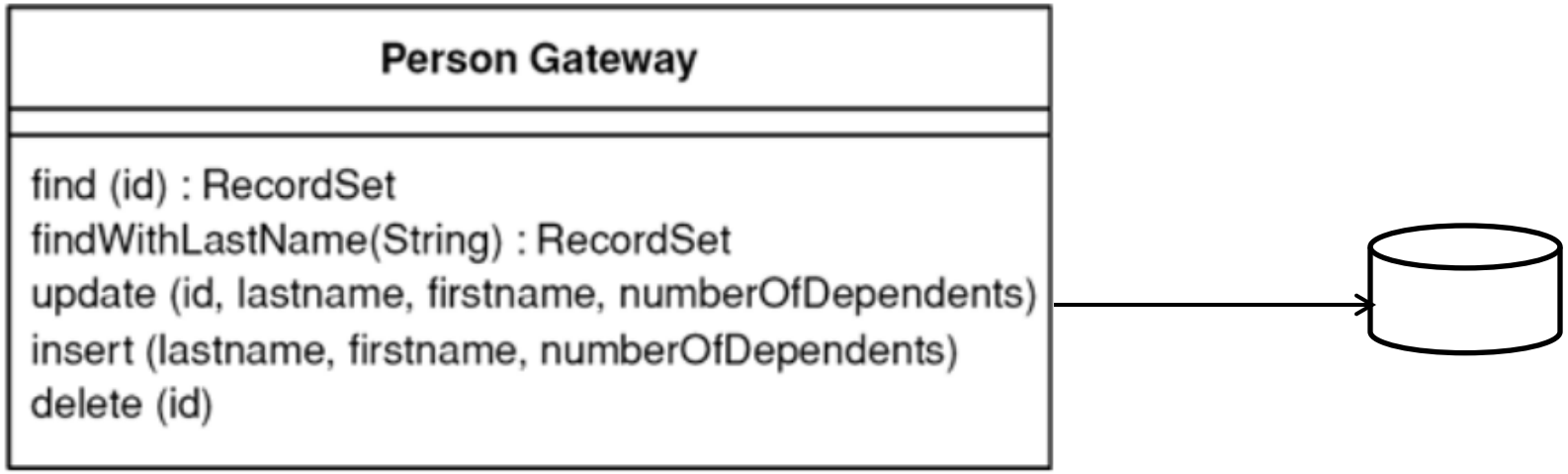
Projektovanje web aplikacija
- uzorci za podatke -

TABLE DATA GATEWAY
(PROLAZ KA TABELI PODATAKA)

Table Data Gateway uzorak

Objekat koji enkapsulira pristup tabeli u bazi podataka (SQL se nalazi samo u njegovoj implementaciji). Jedna instanca upravlja svim redovima u tabeli.



Kako funkcioniše

Table Data Gateway ima jednostavan interfejs, obično se sastoji od nekoliko find metoda koji za dobijanje podataka iz baze i metoda za ažuriranje, umetanje i brisanje. Svaki metod mapira ulazne parametre u SQL upit i izvršava SQL upit nad konekcijom baze.Kapija je obično bez sopstvenog stanja, njena uloga je da prosleđuje podatke u bazu i iz baze.

Primer za Table Data Gateway

```
class OrderGateway {
    private $conn;
    public function __construct(PDO $conn) {
        $this->conn = $conn;
    }
    // CRUD funkcije
    . . .
    // find funkcije
    . . .
}
```

Primer CRUD funkcija

```
class OrderGateway {
    . . .
    public function insert(PDO $conn) {
        $query = 'INSERT INTO orders (reference, amount, vat,
                total, created_at) VALUES (?, ?, ?, ?, ?)';
        $data = array(
                $this->reference, $this->amount, $this->vat,
                $this->total, $this->createdAt->format('Y-m-d H:i:s')
        );
        $stmt = $conn->prepare($query);
        $stmt->execute($data);
        $this->id = $conn->lastInsertId();
    }
}
```

Primer CRUD funkcija

```
class OrderGateway {

    public function update($pk, $ref, $amount, $status) {
            $query = 'UPDATE orders SET reference = ?, amount = ?,
                                    status = ? WHERE id = ?';
            $data = array($ref, $amount, $status, $pk);
            $stmt = $this->conn->prepare($query);
            $stmt->execute($data);
            return $stmt->fetchall();
    }

    public function delete($pk) {
     $stmt=$this->conn->prepare('DELETE FROM orders WHERE id=?');
            return $stmt->execute(array($pk));
    }
. . .
}
```

Upotreba Table Data Gateway objekta

```
$table = new OrderGateway(new PDO('...'));
$table->insert('XX123456789', 358.80, 'unpaid');
$table->update(42, 'XX123456789', 358.80, 'paid');
$table->delete(42);
```

Nastavak primera - Finder funkcije

```
class OrderGateway {
    public function findAll() {
        $query = 'SELECT * FROM orders';
        return $this->conn->query($query)->fetchAll();
    }
    public function find($pk){
        $rs = $this->findBy(array('id' => $pk));
        return 1 === count($rs) ? $rs[0] : false;
    }
    public function findPaidOrders() {
        return $this->findBy(array('status' => 'paid'));
    }
    public function findUnpaidOrders() {
        return $this->findBy(array('status' => 'unpaid'));
    }
}
```

Nastavak – finder funkcije

Za kompleksne upite, primaju se parovi atribut/vrednost i od toga interno konstruiše SQL upit

```
public function findBy(array $criteria) {
    $where = array();
    foreach ($criteria as $field => $value) {
        $where[] = sprintf('%s = ?');
    }
    $q = sprintf('SELECT * FROM orders WHERE %s',
        implode(' AND ', $where));
    $stmt = $conn->prepare($q);
    $stmt->execute(array_values($criteria));
    return $stmt->fetchAll();
}
```

Primer upotrebe finder funkcija

Finder funkcije vraćaju niz redova ili samo jedan red tabele. Red je predstavljen nizom parova ime kolone/vrednost.

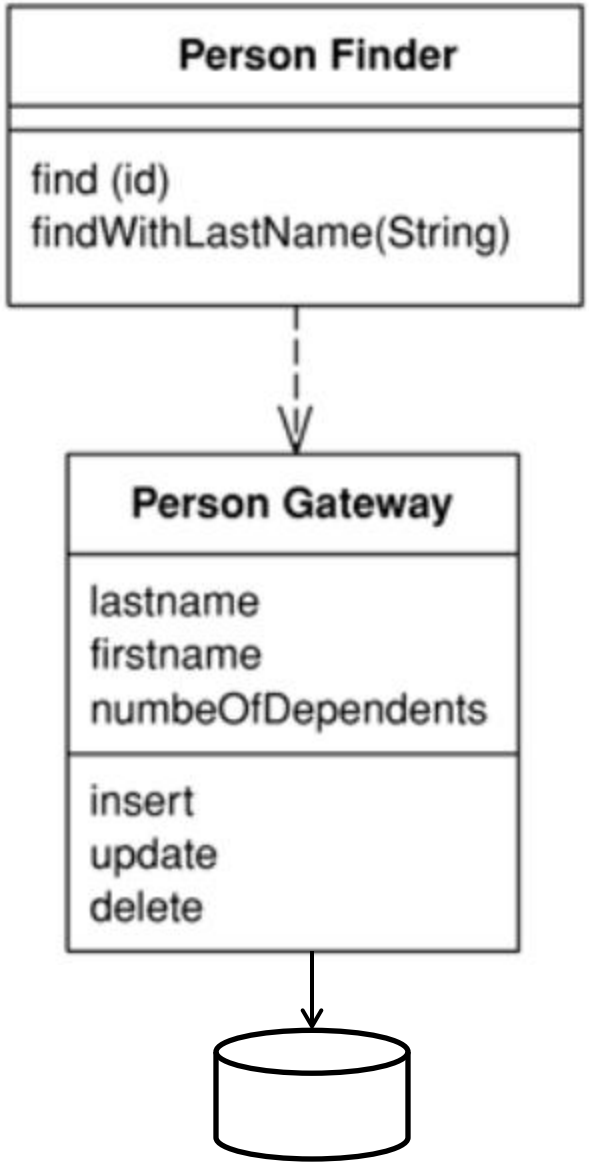
```
$orders = $table->findAll();
$orders = $table->findPaidOrders();
$orders = $table->findUnpaidOrders();
$orders = $table->findBy(array(
'status' => 'paid',
'amount' => 250.00
));
$order = $table->find(42);
$order = $table->findOneBy(array('reference' => '...'));
```

ROW DATA GATEWAY
(PROLAZ KA VRSTI TABELE PODATAKA)

ROW DATA GATEWAY UZORAK

Objekat koji se ponaša kao “prolaz” prema jednom zapisu iz izvora podataka. Postoji jedna instanca po redu.

- Ključna razlika između Table Data Gateway i Row Data Gateway je da prvi reprezentuje čitavu tabelu podataka jednim objektom, dok drugi enkapsulira jedan red tabele jednom instancom gateway objekta. Row Data Gateway može da koristi posebnu Finder klasu, a Table Data Gateway sam implementira finder metod.



Primer implementacije

```
class OrderGateway
{
    private $id; // autoinkrement – interno generisani prim.ključ
    private $reference; // kandidat ključ
    private $amount;
    private $vat;
    private $total;
    private $createdAt;
    // Getters and setters for each property
    // ...
}
```

Ima sopstveno stanje, tj. pamti ključ

Primer CRUD operacija

```
class OrderGateway {
    public function insert(PDO $conn) {
        $query = 'INSERT INTO orders (reference, amount, vat,
                total, created_at) VALUES (?, ?, ?, ?, ?)';
        $data = array(
            $this->reference, $this->amount, $this->vat,
            $this->total, $this->createdAt->format('Y-m-d H:i:s'));
        $stmt = $conn->prepare($query);
        $stmt->execute($data);
        $this->id = $conn->lastInsertId();
    }
}
```

Korišćenje OrderGateway-a

```
$conn = new PDO('...');
$order = new OrderGateway();
$order->setReference('XX12345678');
$order->setAmount(300.00);
$order->setVat(58.80);
$order->setTotal(358.80);
$order->setCreatedAt(new DateTime());
$order->insert($conn);
```

Implementacija klase Finder

```
class OrderFinder {
    static public function findByReference($reference) {
        $query= 'SELECT * FROM orders WHERE reference=?';
        $stmt = $this->conn->prepare($query);
        $stmt->execute(array($reference));
        $rs = $stmt->fetch(); // single row
        return $rs ? OrderGateway::load($rs) : false;
    }
}
```

Finder vraća instancu OrderGateway kao reprezenta traženog reda u tabeli (funkcija load puni objekat iz dobijene kolekcije iz baze).

Implementacija klase Finder

```
class OrderGateway {
    static public function load(array $rs) {
        $order = new OrderGateway($rs['id']); // ovde može da
                                              // se upotrebi Identity Map
        $order->setReference($rs['reference']);
        $order->setAmount($rs['amount']);
        $order->setVat($rs['vat']);
        $order->setTotal($rs['total']);
        $order->setCreatedAt(new DateTime($rs['created_at']));
        return $order;
    }
}
```

Finder koji vraća niz OrderGateway objekata

```
class OrderFinder {
    static public function findMostExpensiveOrders($limit) {
        $orders = array();
        $query = 'SELECT * FROM orders ORDER BY total
                DESC LIMIT ?';
        $stmt = $this->conn->prepare($query);
        $stmt->execute(array($limit));
        $rs = $stmt->fetchAll();

        foreach ($rs as $data) {
                $orders[] = OrderGateway::load($data);
        }
        return $orders;
    }
}
```

Primer upotrebe

```
OrderFinder::setConnection($conn);
$orders = OrderFinder::findMostExpensiveOrders(10);
foreach ($orders as $order) {
    echo $order->getReference(), "\n";
    echo sprintf('%01.2f euros', $order->getTotal()), "\n";
    echo "\n-----\n";
}
```

ACTIVE RECORD
(AKTIVNI ZAPIS)

Active Record uzorak

Objekat koji obuhvata red u tabeli ili pogledu baze podataka, enkapsulira pristup bazi i dodaje domensku logiku na te podatke.

Active Record
=
Row Data Gateway +
Poslovna logika

- Aktivni zapis je veoma sličan kapiji reda podataka RDG. Osnovna razlika je u tome što RDG sadrži samo pristup bazi, dok aktivni zapis sadrži i to i poslovnu logiku (u suštini, razlika je samo formalne prirode). Finder metodi mogu ili da se pridruže samoj ActiveRecord klasi (kada su statički), ili da se odvoje u posebnu Finder klasu, kao kod RDG.

Primer implementacije

- Na klasu OrderGateway iz prethodnog (RDG) primera, možemo dodati neke business metode:

```
class OrderGateway {
    public function applyDiscount($discount) {
        $this->amount -= $discount;
    }
    public function updateTotal() {
        if ($this->vatRate) {
            $this->vat = $this->amount * $this->vatRate;
        }
        $this->total = $this->amount + $this->vat;
    }
}
```

Napomena

- Moguće je uštedeti na pisanju get i set metoda, koristeći osobinu php-a da može dinamički dodeliti polja (osobine) i metode klasi.

```
class Gateway
{
    protected $tableRow;
    public function __set($key, $value) {
        $this->tableRow[$key] = $value;
    }

    public function __get($key) {
        return $this->tableRow[$key];
    }
}
```

- (to se u phpu zove overloading i nema isto značenje kao u drugim jezicima)
- Takođe je moguće neka zajednička polja i metode izvući u superklasu i Gateway klase nasleđivati.

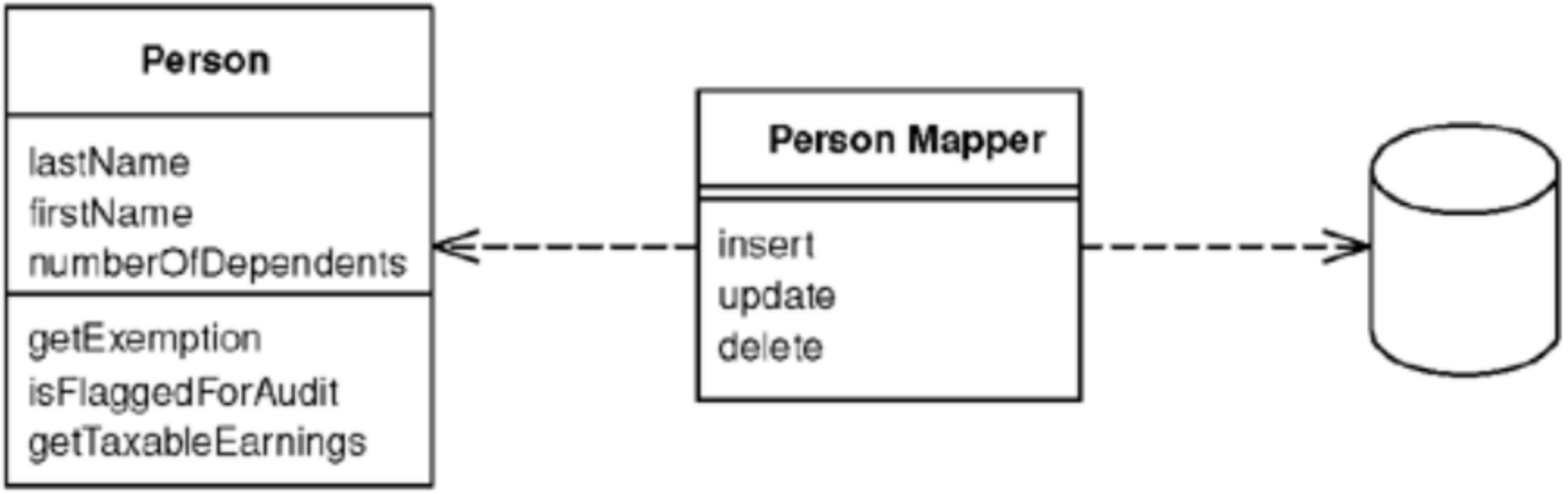
Kada upotrebljavati ove uzorke?

- Prilaz tabeli podataka je verovatno najjednostavniji obrazac za korišćenje od interfejsa ka bazi, jer se lepo preslikava na tabelu baze podataka.
- Aktivni zapis je dobar izbor za domensku logiku koja nije previše složena, sa operacijama kao što su kreiranje, čitanje, ažuriranje i brisanje. Korišćenje podataka i validacije na osnovu jednog zapisa lako se realizuju u ovoj strukturi.
- Glavni problem dosadašnjih uzoraka je u tome što oni dobro rade samo ako domenski objekti odgovaraju direktno tabelama baze podataka, ako su im izomorfne šeme. Ako je, međutim, poslovna logika složena, poželećete da koristite direktno odnose, kolekcije, nasleđivanje itd poslovnih objekata. Ovo se ne mapira lako na Active Record, i parcijalna dodavanja rezultuju u neurednom rešenju.
- U takvoj situaciji preporučeni uzorak je Data Mapper.

DATA MAPPER
(MAPIRANJE PODATAKA)

Data Mapper uzorak (Mapiranje podataka)

Sloj koji omogućava razmenu podataka između objekata i baze održavajući ih nezavisnim jedne od drugih, kao i od samog Mappera.



Objekti i relacione baze podataka imaju različite mehanizme za strukturiranje podataka. Mnogi delovi objekata, kao što su kolekcije i nasleđivanja, nisu prisutni u relacionim bazama podataka. Kada se izgradi objekatni model sa puno poslovne logike, važno je da se mogu koristiti ovi mehanizmi da se bolje organizuju podaci i ponašanje koje ide sa njima.
Na taj način imamo dve različite šeme, to jest, objekatna šema i relaciona šema se ne podudaraju.

Data Mapper

- Pošto treba razmenjivati podatke između dve šeme, to postaje složen problem. Ako objekti u memoriji poznaju relacionu šemu baze podataka, promene u jednoj šemi imaju tendenciju da se preliju i na drugu.
- Data Mapper je posrednički sloj softvera koji razdvaja objekte u memoriji od baze podataka. Njegova odgovornost je prenos podataka između njih i takođe da ih izoluje jedne od drugih.
- Sa Data Mapperom objekti u memoriji ne treba da znaju ni da postoji baza podataka, ne treba im SQL kod, a svakako ni poznavanje šeme baze podataka. (I baza podataka ne zna ništa o objektima koji je koriste). I sam Data Mapper je čak nepoznat domenskim objektima.

“Naivna” implementacija

```
class Order { // domenska klasa
    private $id;
    private $reference;
    private $amount;
    private $vat;
    private $vatRate;
    private $total;
    private $createdAt;
    // Getters and setters for each property
    // ...
}
```

“Naivan” primer Data Mappera

```
class OrderMapper {
    private $conn;
    public function __construct(PDO $conn) {
        $this->conn = $conn;
    }
    public function store(Order $order) {
        // Execute the query to persist the object to the DB
    }
    public function remove(Order $order) {
        // Executes the query to remove the object to the DB
    }
 . . . .
}
```

“Naivna” implementacija Data Mappera

```
class OrderMapper {
    public function findAll()      {
        $objects = array();
        $query = 'SELECT id, reference, vat ... FROM orders';

        foreach ($this->conn->query($query)->fetchAll()  as $data) {
            $object = new Order($data['id']);
            $object->load($data);
            $objects[] = $object;
        }
        return $objects;
    }
    . . . .
}
```

“Naivna” implementacija Data Mappera

```
class OrderMapper {
    public function find($pk) {
        $query= 'SELECT id, vat ...FROM orders WHERE id = ?';
        $object = false;
        $stmt = $this->conn->prepare($query);
        $stmt->execute(array($pk));

        if (false !== $stmt->fetch()) {
            $object = new Order($data['id']);
            $object->load($data);
        }
        return $object;
    }
}
```

Primer upotrebe, kreiranje narudžbe

```
$order = new Order();
$order->setReference('XX12345678');
$order->setAmount(300.00);
$order->setVatRate(0.196);
$order->updateTotal();
$conn = new PDO('mysql:host=localhost ...');
$mapper = new OrderMapper($conn);
$mapper->store($order);
```

Primer upotrebe, pretraga i ažuriranje

```
$conn = new PDO('mysql:host=localhost ...');
$mapper = new OrderMapper($conn);
$order = $mapper->find(42);
$order->setAmount(399.00);
$order->updateTotal();
$mapper->store($order);
```

Jedan veliki problem sa našim finderima

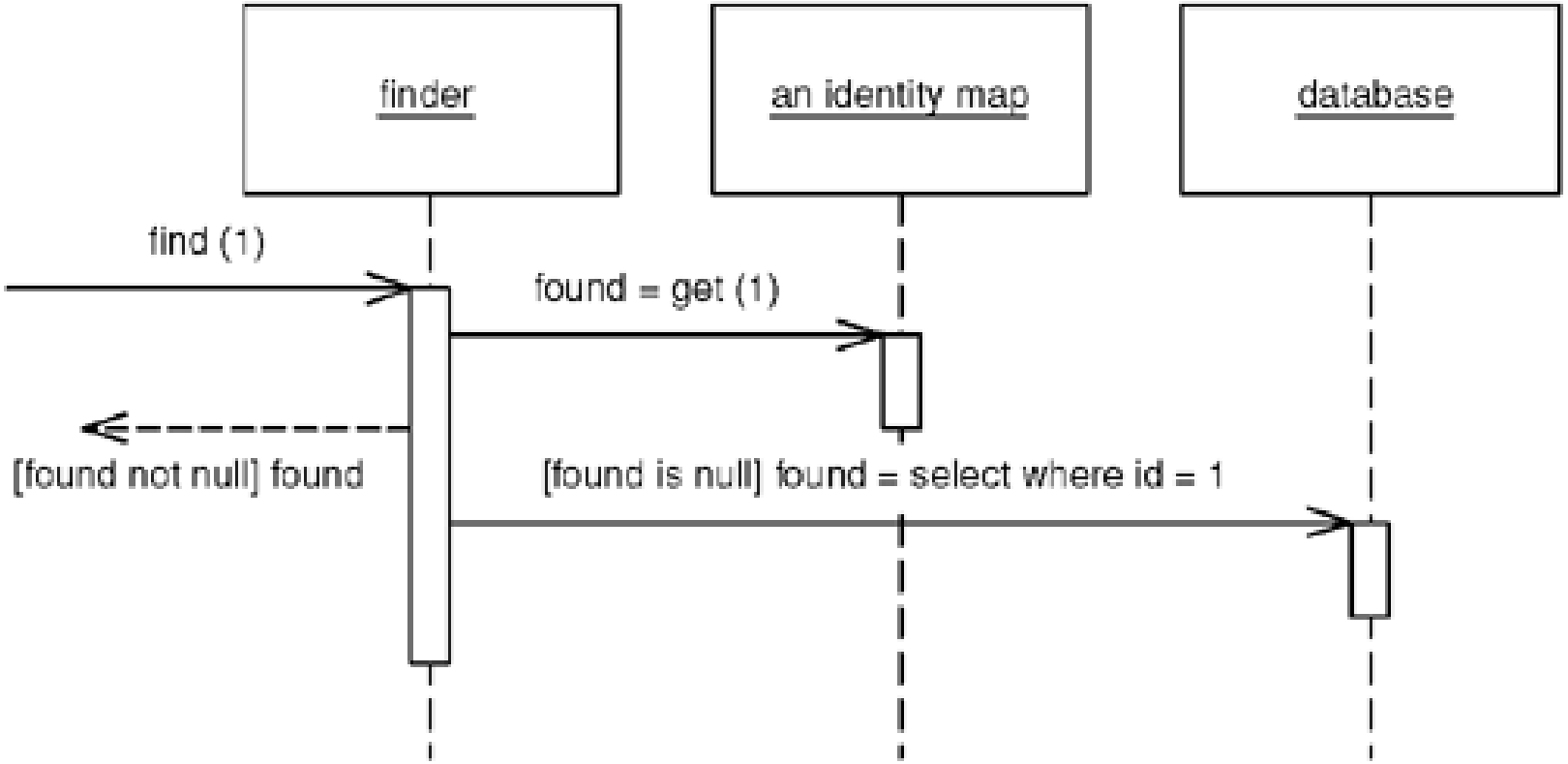
```
$conn = new PDO('mysql:host=localhost ...');
$mapper = new OrderMapper($conn);
$order = $mapper->find(42);
$order->setAmount(399.00); // promenjen je objekat u memoriji
$order2 = $mapper->find(42); // čita inicijalne podatke iz baze
$order2->updateTotal(); // nema promene
$mapper->store($order2); // novi Amount je zaboravljen
```

- Dakle problemi konzistencije podataka nastaju kada možemo napraviti različita dva memorijska objekta za isti entitet u bazi. Rešenje na nivou obrade kompletnog zahteva jednog korisnika je uzorak **Identity Map**.
- Kada se uzme u obzir konkurentni pristup bazi od strane više korisnika, jedno rešenje je eksplicitno koristiti database transakcije u kodu koji koristi domenske objekte.
- Alternativno, može se transakciona obrada ugraditi u Data Mapper kod putem **Unit of Work** patterna.

IDENTITY MAP
(MAPA IDENTITETA)

Identity Map uzorak

- Obezbeđuje da svaki objekat bude učitan samo jednom držeći svaki učitani objekat u mapi. Traži objekte uz pomoć mape kada se referiše na njih.



Skica implementacije Identity Map-a

```
class IdentityMap {
private $entities;
    public function fetch($class, $pk) {
        $key = $this->getKey($class, $pk);
        if (isset($this->entities[$key]))
            return $this->entities[$key];
        return false;
    }
    public function store($entity) { // domain objekat, npr. order
        $key = $this->getKey($class, $entity->getId());
        $this->entities[$key] = $entity;
    }
    private function getKey($class, $pk) {
        return $class.'-'.$pk;
    }
}
```

Dodavanje Identity Map-a u Data Mapper

```
class OrderMapper {
    private $map;
    public function __construct(IdentityMap $map, ...) {
        $this->map = $map;
        . . .
    }
    public function store(Order $order) {
        // ovde treba upisati podatke iz $order u bazu
        $this->map->store('Order', $order);
    }
    public function find($pk) {
        if (false !== $order = $this->map->fetch($pk))
            return $order;
        // čitanje iz baze i kreiranje novog $order
        $this->map->store('Order', $order);
        return $object;
    }
}
```

Eksplicitno korišćenje transakcija

(Primer iz Data Mapper-a za CodeIgniter aplikativni okvir)

```
// Create user
$u = new User(); // domain object

// Populate with form data
$u->username = $this->input->post('username');
$u->email = $this->input->post('email');
$u->password = $this->input->post('password');
$u->confirm_password = $this->input->post('confirm_password');

// Begin transaction
$u->trans_begin();

// Attempt to save user
$u->save();
. . . . .
```

Eksplicitno korišćenje transakcija

```
// Check status of transaction
if ($u->trans_status() === FALSE) {
    // Transaction failed, rollback
    $u->trans_rollback();
    // Add error message
    $u->error_message('transaction', 'The transaction failed to save (insert)');
}
else {
    // Transaction successful, commit
    $u->trans_commit();
}

// Show all errors
echo $u->error->string;
// Or just show the transaction error we manually added
echo $u->error->transaction;
```

Alternativno korišćenje

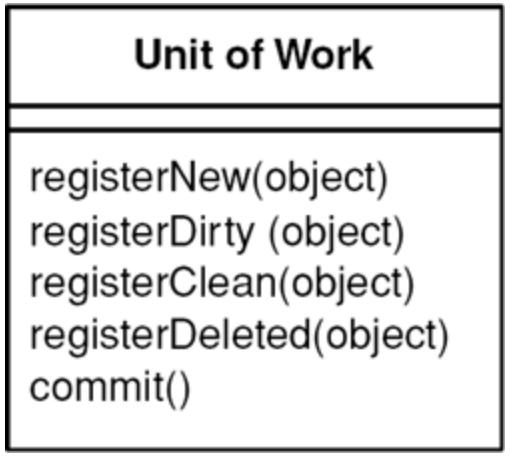
- Malo je jednostavnije za korisnika Data Mappera ako se transakciona obrada ubaci u save i update metode:

```
// Create user
$u = new User();
// Populate with form data
 . . . .
// Attempt to save user
if ($u->save()) {
    // Saved successfully
}
else {
    // Show all errors
    echo $u->error->string;
    // Or just show the transaction error
    echo $u->error->transaction;
}
```

UNIT OF WORK
(JEDINICA RADA)

Uzorak Unit of Work

- Održava listu objekata uključenih u poslovne transakcije i koordinira upisivanje promena i rešava probleme konkurencije.
- Kada vučete podatke iz baze podataka, važno je da pratite šta ste menjali; U suprotnom, podaci neće biti zapisani nazad u bazu. Slično morate da ubacite nove objekte koje ste kreirali i uklonite sve objekte koje ste obrisali.
- Možete da menjate u bazi podataka sa svakom promene vašeg objektnog modela, ali to može da dovede do mnogo malih pristupa bazi, koji se sporo izavršavaju. Dodatno je potrebno da imate otvorenu transakciju tokom cele interakcije, što je nepraktično ako imate poslovnu transakciju koja obuhvata više zahteva.
- Situacija je još gora ako treba da pratite objekte koje ste pročitali, da bi izbegli nedoslednosti.
- Jedinica rada vodi evidenciju o svemu što radite tokom poslovne transakcije što može da utiče na bazu podataka. Kada završite, ona preduzme sve što treba da se uradi da promeni bazu podataka kao rezultat vašeg rada.

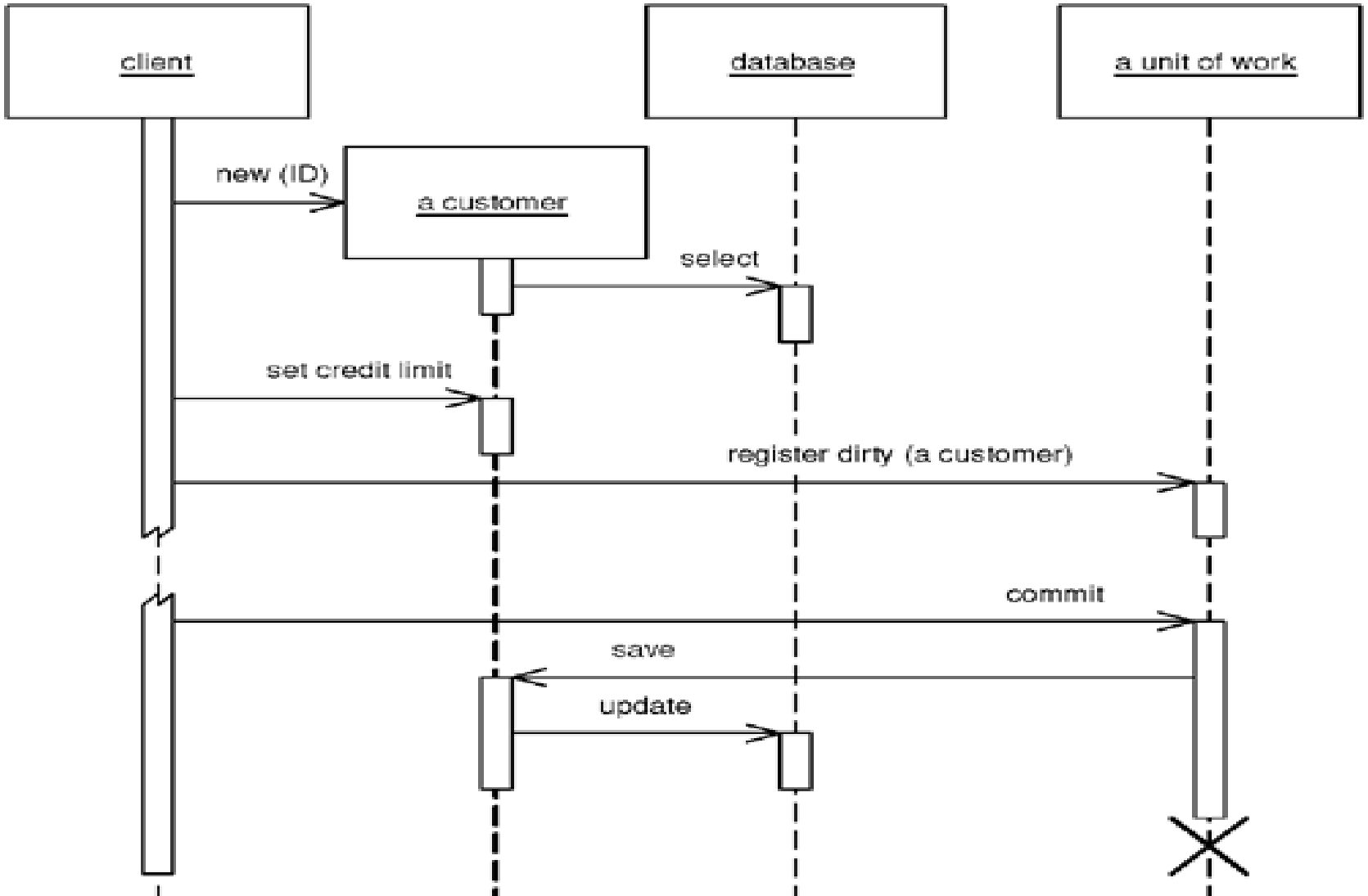


Kako funkcioniše Jedinica rada

- Svaki put kada se kreira, menja ili briše objekat to se saopšti jedinici rada. Ona takođe zna o objektima koji su učitani iz baze tako da može da proveri nekonzistencije u čitanju tako što proverava da nijedan od objekata nije promenjen u bazi podataka u toku poslovne transakcija.
- Ključna stvar o jedinici rada je da, kada dođe vreme da se izvrši commit, jedinica rada odlučuje šta da radi. Ona otvara transakciju, radi kontrolu konkurentnosti (koristi pesimističko ili optimističko zaključavanje), i zapisuje promene u bazi podataka.
- Programer nikada eksplicitno ne poziva metode ažuriranja baze. Na taj način on ne mora da prati šta se promenilo ili da brine o tome kako referencijalni integritet utiče na poredak u kome treba da se rade stvari.
- Programer (ili sam poslovni objekat) mora jedinici rada da saopšti o kojim objektima treba da vodi računa.

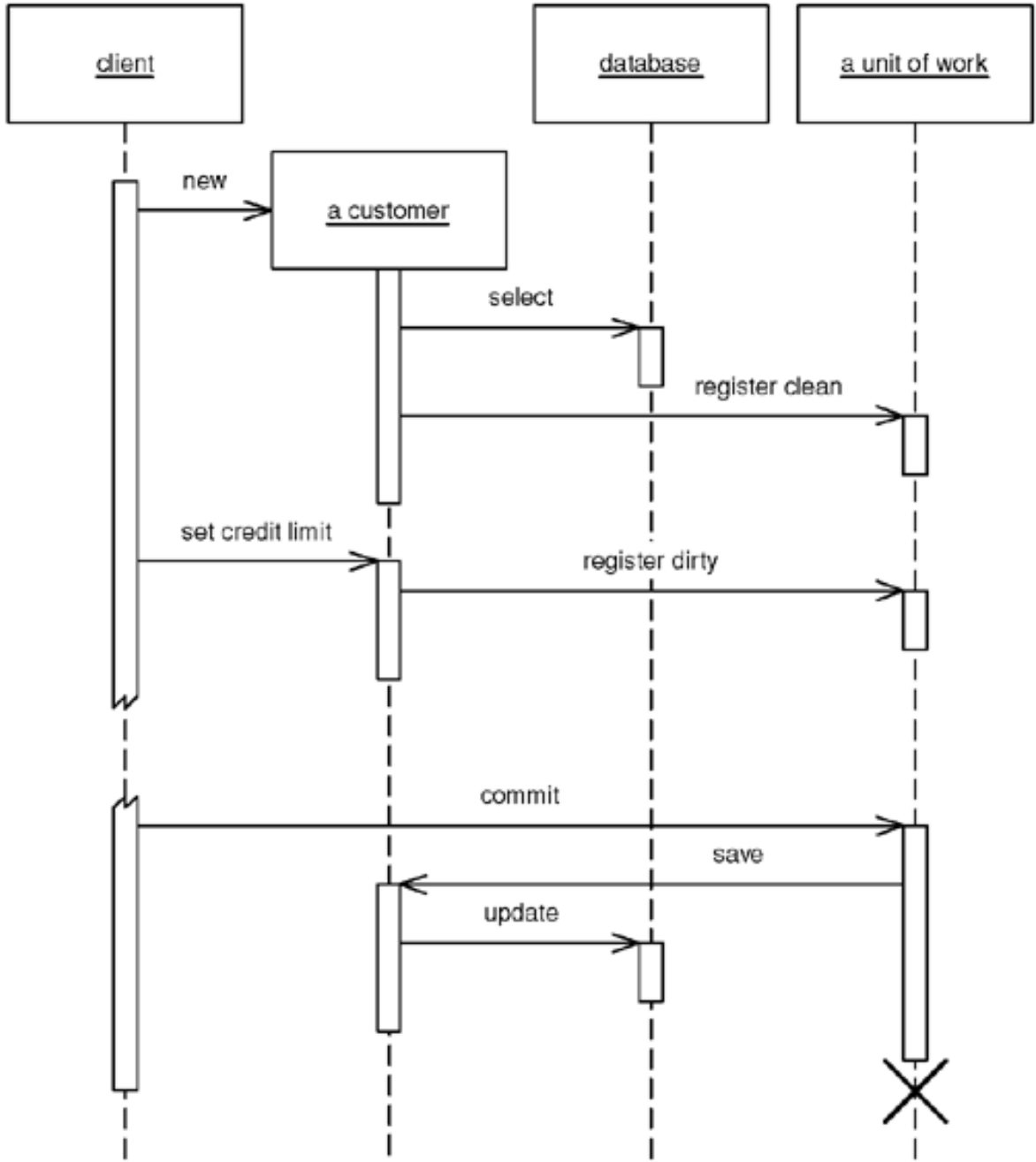
Pozivalac registruje promenjeni objekat

- Sa registracijom od strane pozivaoca, korisnik objekta mora registrovati objekat za promene kod Jedinice rada. Objekat koji nije registrovan neće biti upisan pri commitu.



Objekat sam sebe registruje

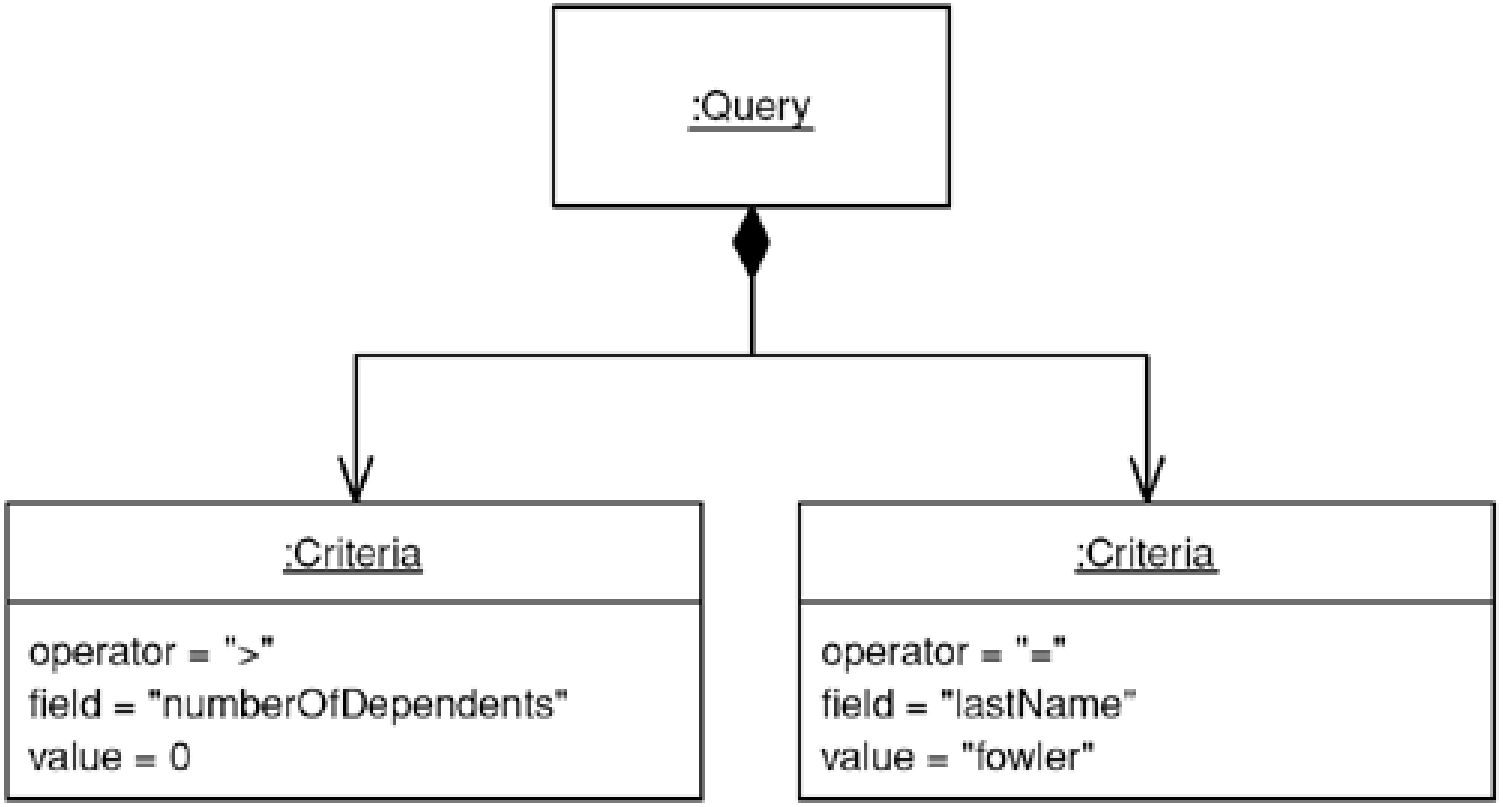
- Registracija se vrši u metodama objekta.
- Učitavanje objekta iz baze registruje objekat kao čist;
- setXY metode registruju objekat kao prljav.
- Jedinica za rad treba da bude prosleđena objektu ili da bude na nekom dobro poznatom mestu.



QUERY OBJECT
OBJEKAT UPITA

Uzorak Query Object

- Objekat koji predstavlja upit u bazu podataka



Upit Objekat je GoF interpreter, to jest, struktura objekata koji mogu da formiraju SQL upit. Upit može da se odnosi na objekte i polja a ne na tabela i kolona. Na ovaj način, oni koji pišu upite mogu to činiti nezavisno od šeme baze i promena u šemi može biti lokalizovana na jednom mestu.

Primer upotrebe

```
$query = Query::create()
->select(array('id', 'reference', 'amount', 'status'))
->from('orders')
->where(Criteria::equals('status', 'paid'))
->where(Criteria::greaterThan('amount', 2000))
->where(Criteria::like('reference', 'XX123%'))
->orderBy('amount', 'desc')
->getSql()
;
// SELECT id, reference, amount, status
// WHERE status = ? AND amount > ? AND reference LIKE ?
// ORDER BY amount DESC
```

Elementi implementacije uzorka Query Object

```
class Criteria {
    private $field;
    private $operator;
    private $parameters;
    public function __construct($field, $operator, $value) {
        $this->field = $field;
        $this->operator = $operator;
        $this->parameters[] = $value;
    }
    static public function equal($field, $value, $vars) {
        return new Criteria($field, '=', $vars);
    }
    static public function notEqual($field, $value, $vars) {
        return new Criteria($field, '<>', $vars);
    }
}
```

Kada koristiti Query Object

- Upitni objekti često nisu neophodni, pošto su mnogi programeri zadovoljni SQLom. Sa njim se mogu sakriti mnogi detalje o šemi baze podataka iza određene strukture sa metodama.
- Moguće ga je koristiti sa Data Mapperom
- Prednosti objekta upita ispoljavaju se kod sofisticiranih potreba: enkapsuliranje šeme baze, podrška za više baza podataka, podrška za višestruke šeme i optimizacija upita da se izbegnu višestrustruki upiti.

Zaključak

- Razumevanje OO dizajna web aplikacija zasnovano je na poznavanju projektnih uzoraka koji se primenjuju u njihovom projektovanju.
- Definisali smo najvažije uzorke i ilustrovali implementacije opšte strukture web aplikacije, kontrolnog, prezentacionog sloja i sloja modelovaja podataka.
- Ne treba samostalno implementirati pomenute šablone, nego iskoristiti gotove biblioteke i aplikativne okvire (frameworks), koji su popularni i provereno ispravno implementirani.