The PHP Company

Modeling With Zend\Db

As of Zend Framework 2.0.*

Who Am I?

- **Ralph Schindler (ralphschindler)**
  - Software Engineer on the Zend Framework team
    - At Zend for 5 years
    - Before that TippingPoint/3Com
  - Programming PHP for 13+ years
  - Live in New Orleans, LA.
    - Lived in Austin, Tx for 5 years

This Webinar

- Brief words on “Modeling”
- Patterns For Modeling
- Zend\Db’s as a tool in Modeling
- Look at a real application:
  - https://github.com/ralphschindler/PatternsTutorialApp

What Is Modeling?

- **(Loosely defined) M in the MVC**
  - http://en.wikipedia.org/wiki/Model-view-controller
- **From Wikipedia:**
  - A controller can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can send commands to the model to update the model's state (e.g., editing a document).
  - A **model** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.
  - *A view requests from the model the information that it needs to generate an output representation.*

What does that mean really?

- **In PHP, you can generally think of it like this:**
  - Controllers interact with environment
    - $_POST, $_SERVER, $_GET, environment variables, etc
  - Views are responsible for display concerns
    - What does my HTML look like
    - As I iterate this object or array, how do I format it
    - How do I escape data for consumption in a web browser
  - Which leaves the Model...

The Model is ...

- **A set of characteristics:**
  - The core of your business problem
  - Data & the persistence of that data
  - UI agnostic (HTML and Json agnostic)
    - aka: View agnostic concerns / Not a view
    - Models don't have an opinion on how they are displayed
  - Environment agnostic (CLI vs. Browser)
    - aka: Controller agnostic concerns / Not a controller
    - Models don't have an opinion on how they are consumed
    - ...

... continued,

- In OO terms:
  - OOM: Object oriented modeling
  - A way of conceptualizing a problem domain into classes and objects to better manage their complexity, to simplify it
  - Present business object workflows in easy to understand and consume API

How do we build an API?

- **We could just interact with the datasource directly**
  - This offers little abstraction and leaves us with a persistence centric API
- **We need to find a suitable level of abstraction**
  - For this we need patterns...

Patterns: The tools in our toolbox

- **Different patterns describe a particular abstraction, that might suit our need**
- **Ones we'll cover:**
  - TableGateway, RowGateway
    - Implemented by Zend\Db
  - ActiveRecord
  - Mapper
  - Lazy Loading & Lazy Loading Via PHP Closure
- **Domain Driven Design patterns:**
  - Repository
  - Entity, Value Object, Value
  - Other briefly for context

TableGateway & RowGateway

- **Implemented in Zend\Db**
- **TableGateway, specifically, can be used:**
  - Directly as the gateway to “model data”
    - No abstraction: “Models” are really associative arrays in this scenario
  - Directly as the data access for a Repository
    - 1 level of abstraction: Essentially as a mapper
  - Or as the implementation detail of a Mapper
    - 2 levels of abstraction: Repository > Mapper > TableGateway

TableGateway & RowGateway

```php
interface TableGatewayInterface
{
    public function getTable();
    public function select($where = null);
    public function insert($set);
    public function update($set, $where = null);
    public function delete($where);
}
```

```php
interface RowGatewayInterface
{
    public function save();
    public function delete();
}
```

ActiveRecord

```php
class ActiveRecord
{
    public static function findBy($where) {}
    public static function __callStatic($method, $args) {}

    public function __construct($table, $rowData) {}
    public function save();
    public function delete();
    public function __call($method, $args) {}
    public function __get($name) {}
    public function __set($name, $value) {}
}
```

Mapper

```php
class ArtistMapper
{
    public function mapArrayToArtist(array $data, Artist $artist = null)
    {
        $artist = ($artist) ?: new Artist;
        $artist->firstName = $data['first_name'];
        $artist->lastName  = $data['last_name'];

        $album = new Album;
        $album->title = $data['album_1_title'];
        $artist->albums[] = $album;
        return $artist;
    }
}
```

Lazy Loading Via Closure/Anon Func.

```php
class DataMapper {
    protected function mapPlaylistRowToObject(array $row) {
        $playlist = new Playlist;
        $playlist->setId($row['id']);
        $playlist->setName($row['name']);
        $playlist->setTracks($this->lazyLoadTracksClosure($row['id']));
        return $playlist;
    }

    protected function lazyLoadTracksClosure($playlistId) {
        $dataMapper = $this; // php 5.3 hack, must be renamed
        return function () use ($playlistId, $dataMapper) {
            // ...
            return $tracks;
        }
    }
}
```

Lazy Loading Via Closure/Anon Func.

```php
class Playlist {
    public function setTracks($tracks) {
        $this->tracks = $tracks;
    }
    public function getTracks() {
        if ($this->tracks instanceof \Closure) {
            $this->tracks = call_user_func($this->tracks);
        }
        return $this->tracks;
    }
}
```

Repository

- "Persistence Ignorance" is the idea that at a particular level of your abstraction, the API knows nothing about (*the details*) how something is persisted
- Implementations of a Repository can deal with persistence, but this should not be exposed in the API of this class (or the interface for the Repository)

Repository

```php
interface TrackRepositoryInterface {
    // @return Track[]
    public function findAll();
    public function findById($id);

    public function store(Track $track);
    public function remove(Track $track);
}
```

```php
// IMPLEMENTATION
class DbTrackRepository implements TrackRepositoryInterface {
    public function __construct(TrackDbMapper $mapper) {}
    /** ... **/
}

// USAGE
$trackRepo = new DbTrackRepository($services->get('TrackMapper'));
$tracks = $trackRepo->findAll();
foreach ($tracks as $track) {
    // do something interesting
}
```

Entity, Value Object, Value

- An *Entity* has an identity and a *value object* does not.
- Both are generally POPO's (Plain old PHP objects).
- By definition, *value objects* are identity free and immutable.
- *Values* are simply put, any scalar in PHP (for all intents and purposes).
- Two separate *Entities* can share the same reference to a *Value Object*.

Entity

```php
class Artist {
    public $id;  // has identity!
    public $name; // has identity!
    public $yearFormed;
}
```

Value Object

- PHP's DateTime object does not qualify:

```
class DateTime {
    public function modify(/* string */ $modify);
    public function setSomething();
    public function add(DateInterval $interval);
}
```

- Your own will:

```
class Date {
    public getYear();
    public getMonth();
    public getDay();
    public function __get($name);
}
```

Others: Layered Architecture

- **Layered Architecture**
  - A way of dividing out software conceptually
  - In PHP, this might happen with some usage of namespaces
  - The type of pattern it implements implies the layer of code it belongs to

Other: Services

- **Services**
  - Service Layer: separate abstraction layer between controllers and models
  - Model Services: (DDD) A place where "workflows/functions that have no natural place in a value object/entity"
  - Dependency Injection / Application Architecture: shared objects, dependencies (Service Locator)

Other: Aggregate & Aggregate Root

- A Domain Driven Design Term
- Aggregate: the series of objects in a model bound together by references and associations
- Aggregate Root: Only object outside members can hold a reference to, the "entry object", the primary object

So you want to build an app?

- **Most modern and stable frameworks (full stack & micro) give you an option on how to handle "persistence" in models**
  - ZF has Zend\Db, but no Zend\Model
  - ZF has a user contributed module for Doctrine integration
  - Slim, Silex, etc. don't ship with any persistence solution
  - Symfony 2 ships with tools to integrate Doctrine
- **Early frameworks shipped an ActiveRecord-like solution**
- **Persistence is not always a database**
  - Could be a web service
  - Could be a document database

Zend\Db In Modeling

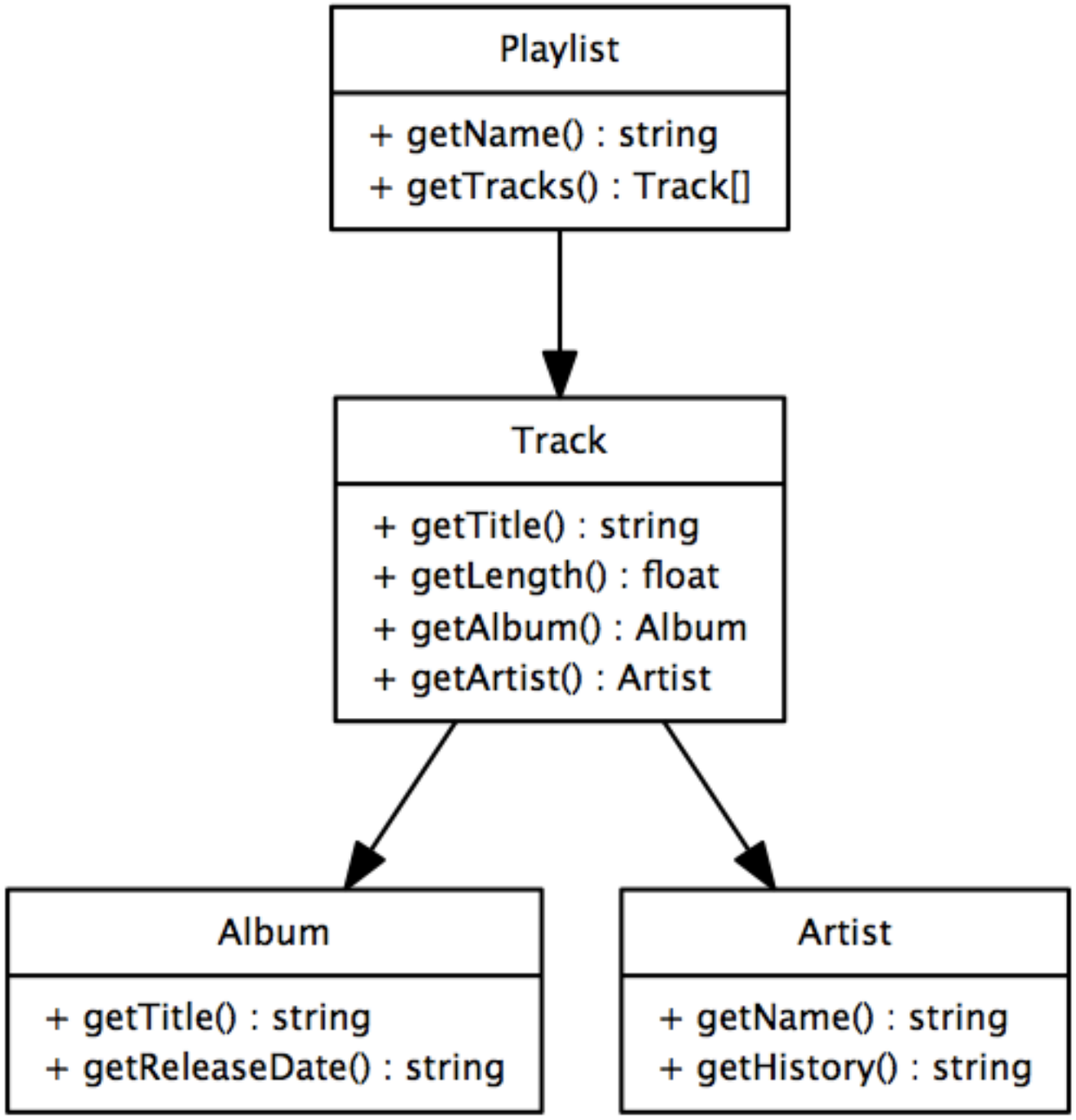
- **What does one *need to know*?**
  - Previous webinar:
    - Have an overall idea of the architecture:
      - Zend\Db\Adapter's Drivers & Platform objects for Driver Abstraction
      - Zend\Db\Sql for Sql as OO **as well as** SQL abstraction
  - Strengths of Zend\Db
    - The base TableGateway is a solid approach to an object per table
    - The Zend\Db\Sql\Select API is expansive and offers full a framework for full SQL abstraction
    - Zend\Db is not a *modeling framework* on its own
      - Doctrine is a better solution for this

Zend\Db\Sql\Select

- Let's have a look at some example queries to get a feel for the Select API
  - https://gist.github.com/3949548

Let’s build an app!

- **Code Location:**
  - https://github.com/ralphschindler/PatternsTutorialApp/
- **Problem Domain:**
  - I there is money in sharing playlists online.
  - I am not sure what the business will be, but I know it centers around a playlist
  - We need to be able to model Track, Arist and Album information
  - We might want to be able to pull information from web services


Playlist
+ getName() : string
+ getTracks() : Track[]
Track
+ getTitle() : string
+ getLength() : float
+ getAlbum() : Album
+ getArtist() : Artist
Album
+ getTitle() : string
+ getReleaseDate() : string
Artist
+ getName() : string
+ getHistory() : string

Thanks!

http://twitter.com/ralphschindler

http://framework.zend.com/zf2

http://github.com/zendframework/

http://github.com/ralphschindler