OOP

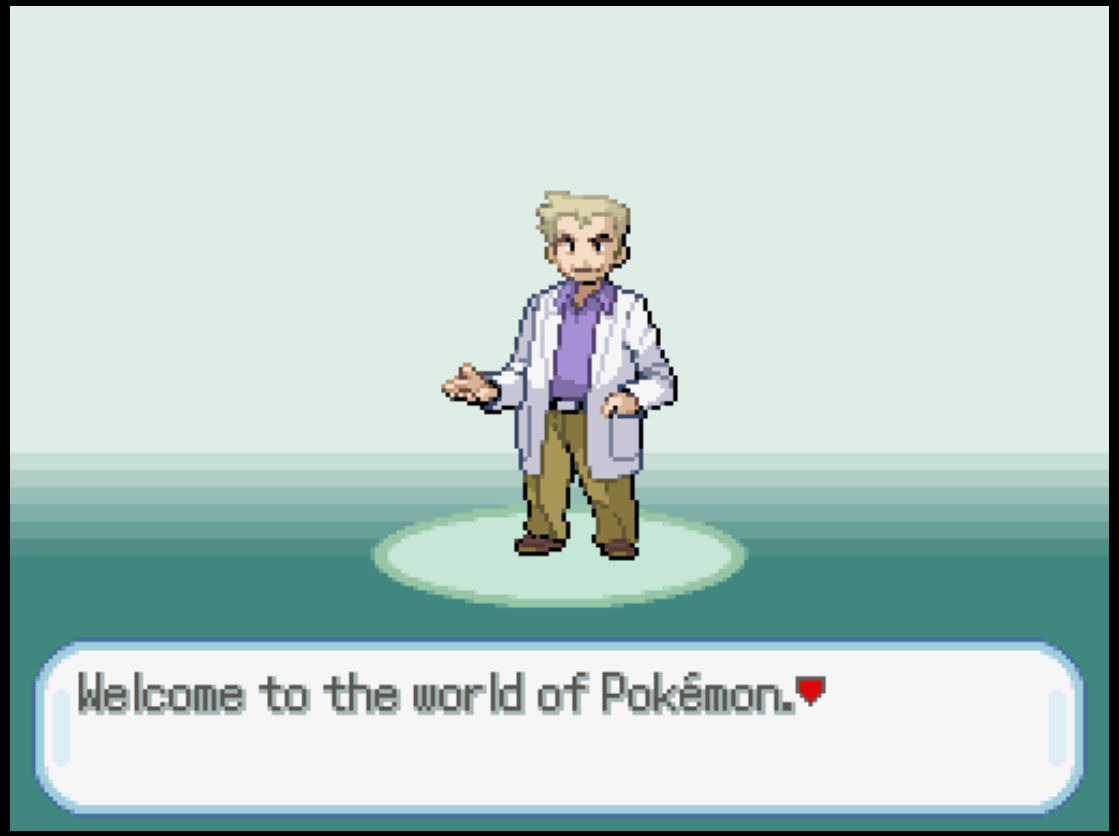
The Pokemon Journey

Hi I'm Fatima

@sugaroverflow

- Digital Echidna
- Drupal Diversity & Inclusion

The Journey Begins

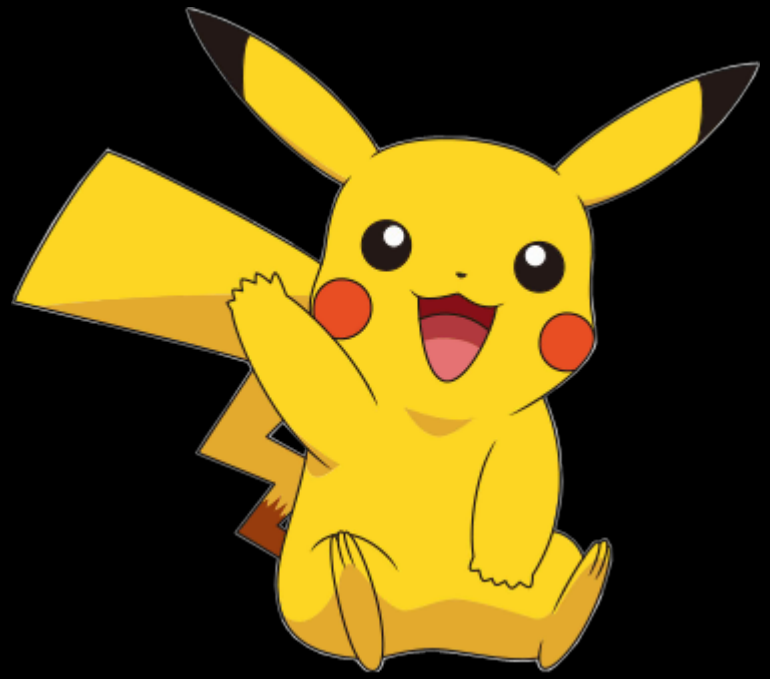


Our first Pokemon!

Pikachu

type: electric

attack: thunderbolt



Classes are like blueprints

Properties

data points

Methods

class-specific functions

Creating a Pokemon Class

```
class Pokemon {
  public $name;
  public $type;
}
```

Creating a Pokemon Class

```
class Pokemon {
  public $name;
  public $type;
}
```

```
class Pokemon {
  public $name;
  public $type;

  public function attack() {
    /* do something */
    return $attack;
  }
}
```

Creating a Pokemon Class

```
class Pokemon {
  public $name;
  public $type;
}
```

```
class Pokemon {
  public $name;
  public $type;

  public function attack() {
    /* do something */
    return $attack;
  }
}
```

```
public function sleep($time_asleep) { ... }
```

Constructing a Pokemon

PHP provides magic method: `__construct()`

```
class Pokemon {
  public $name;
  public $type;

  public function attack() {
    /* do stuff */
    return $attack_stuff;
  }

  public function __construct($name, $type) {
      $this->name = $name;
      $this->type = $type;
  }
}
```

Constructing a Pokemon

PHP provides magic method: `__construct()`

```
class Pokemon {
  public $name;
  public $type;

  public function attack() {
    /* do stuff */
    return $attack_stuff;
  }

  public function __construct($name, $type) {
      $this->name = $name;
      $this->type = $type;
  }
}
```

an *object* is an *instance* of a *class*

Creating a Pokemon Object

```
public function __construct($name, $type) { ... }
```

Creating a Pokemon Object

```
public function __construct($name, $type) { ... }
```

Instantiating a Pokemon object:

```
$pikachu = new Pokemon('Pikachu', 'Electric');
```

Creating a Pokemon Object

```
public function __construct($name, $type) { ... }
```

Instantiating a Pokemon object:

```
$pikachu = new Pokemon('Pikachu', 'Electric');
```

Calling a method from the class:

```
$pikachu->attack();
```

Classes in Drupal (example)

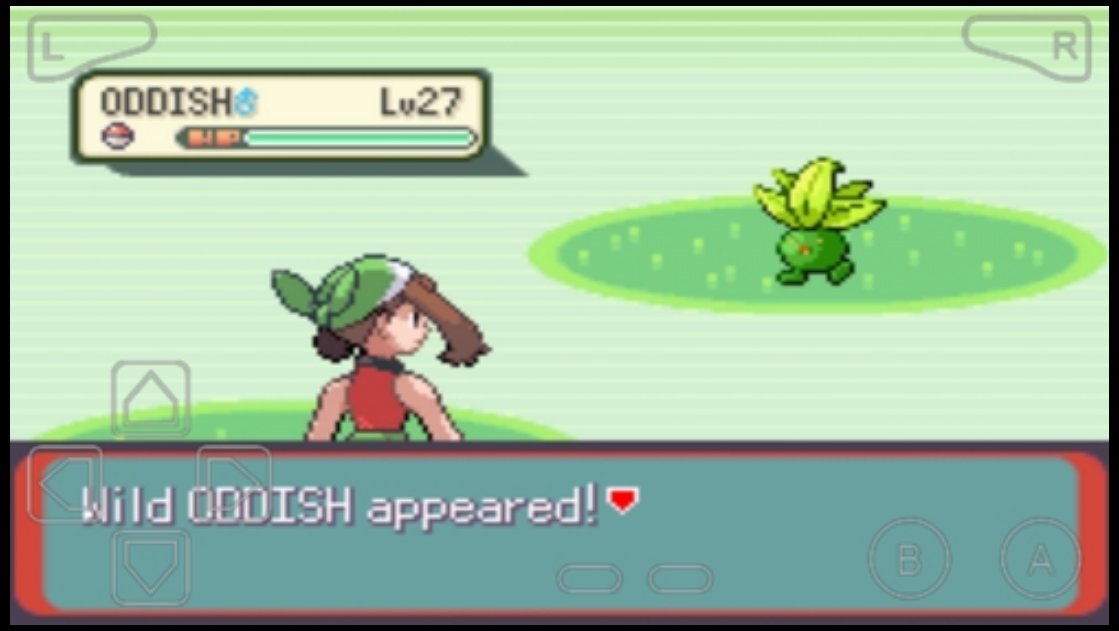
```
class Link {
  public function __construct($text, Url $url) {
    $this->text = $text;
    $this->url = $url;
  }
}
```

Classes in Drupal (example)

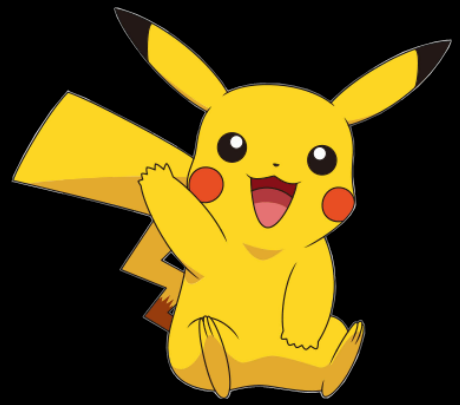
```
class Link {
  public function __construct($text, Url $url) {
    $this->text = $text;
    $this->url = $url;
  }
}
```

```
abstract class Entity implements EntityInterface {
  public function toLink($text = NULL, $rel = 'canonical', array $options = []) {
    // do stuff to variables
    return new Link($text, $url);
  }
}
```

A Wild Pokemon Appears!



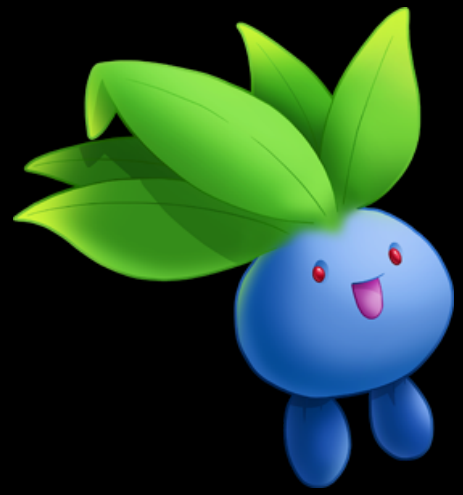
Our Pokemon



Pikachu

type: Electric

attack: Thunderbolt



Oddish

type: Grass

attack: Poison Fire

Inheritance is about sharing

parent class

child classes

- inherit methods from the parent class
- can override methods or properties

Inheritance is about sharing

parent class

child classes

- inherit methods from the parent class
- can override methods or properties

PHP is a single inheritance language

Pokemon Inheritance

```php
class Pokemon {
  public $name;
  public $type;

  public function attack() {
    /* do stuff */
    return $attack_stuff;
  }

  public function __construct($name, $type) {
      $this->name = $name;
      $this->type = $type;
  }
}
```

Pokemon Inheritance

```php
class Pokemon {
  public $name;
  public $type;

  public function attack() {
    /* do stuff */
    return $attack_stuff;
  }

  public function __construct($name, $type) {
      $this->name = $name;
      $this->type = $type;
  }
}
```

```php
class ElectricPokemon extends Pokemon { ... }
```

Extending the Pokemon class

```
class ElectricPokemon extends Pokemon {
  public $type = "Electric";
}
```

Extending the Pokemon class

```
class ElectricPokemon extends Pokemon {
  public $type = "Electric";
}
```

```
class Pikachu extends ElectricPokemon {
  public $name = "Pikachu";

  public function attack() {
    return 'Thunderbolt!';
  }
}
```

Extending the Pokemon class

```
class ElectricPokemon extends Pokemon {
  public $type = "Electric";
}
```

```
class Pikachu extends ElectricPokemon {
  public $name = "Pikachu";

  public function attack() {
    return 'Thunderbolt!';
  }
}
```

```
$pikachu = new Pikachu();
$pikachu->attack(); // "Thunderbolt!"
```

Inheritance in Drupal

```
interface WidgetInterface extends WidgetBaseInterface {
   public function formElement(...);
}
```

```
abstract class WidgetBase implements WidgetInterface { }
```

Inheritance in Drupal

```
interface WidgetInterface extends WidgetBaseInterface {
   public function formElement(...);
}
```

```
abstract class WidgetBase implements WidgetInterface { }
```

```
class RangeWidget extends WidgetBase {
  public function formElement(FieldItemListInterface...) {

    $element['from'] = [ ... ]
    $element['to'] = [ ... ]
  }
}
```

Visibility

public

protected

private

Visibility

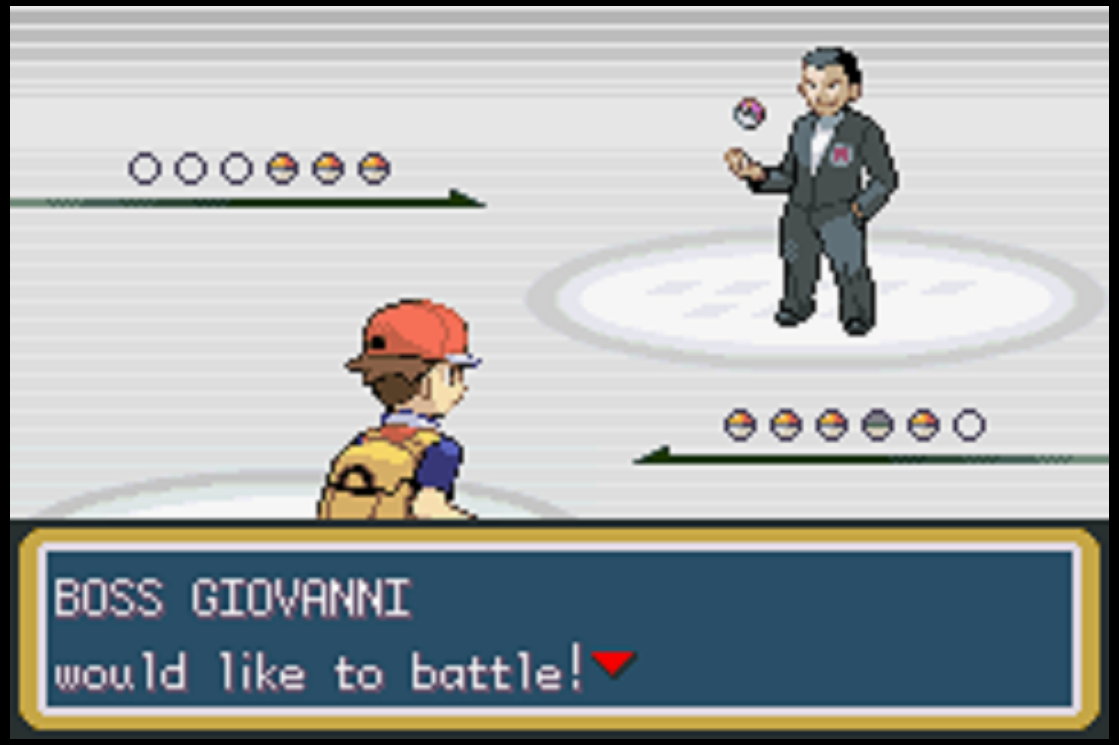
public

protected

private

child classes inherit all the *public* or *protected* properties of their parent class.

Gym Battle!



Comparing our Pokemon

Pikachu

type: Electric

attack: Thunderbolt

strength: Water

weakness: Ground

Oddish

type: Grass

attack: Poison Fire

strength: Water

weakness: Fire

Getting data from our Pokemon

```
class Pokemon {
  // properties
  // attack() method
  // constructor

  protected $weakness;
  protected $strength;

  public function getWeakness() {
    /* do stuff */
    return $this->weakness;
  }

  public function setStrength($strength) {
    /* do stuff */
    $this->strength;
  }
}
```

In child classes

```
class ElectricPokemon extends Pokemon {
  protected $strength = 'grass';
  protected $weakness = 'water';
  // other properties and methods..
}
```

In child classes

```
class ElectricPokemon extends Pokemon {
  protected $strength = 'grass';
  protected $weakness = 'water';
  // other properties and methods..
}
```

```
class Pikachu extends ElectricPokemon {
  // properties and attack() method.
}
```

In child classes

```php
class ElectricPokemon extends Pokemon {
  protected $strength = 'grass';
  protected $weakness = 'water';
  // other properties and methods..
}
```

```php
class Pikachu extends ElectricPokemon {
  // properties and attack() method.
}
```

```php

$pikachu = new Pikachu();
$pikachu->getWeakness(); // returns 'water';
```

Getters/Setters in Drupal

```
class RouteMatch implements RouteMatchInterface {
  public function getParameters() {
    return $this->parameters;
  }
}
```

Getters/Setters in Drupal

```
class RouteMatch implements RouteMatchInterface {
  public function getParameters() {
    return $this->parameters;
  }
}
```

```
$params = \Drupal::routeMatch()->getParameters();
```

Pokemon classes

```
Class Pokemon
```

```
Class Electric Pokemon extends Pokemon
```

```
Class Pikachu extends ElectricPokemon
```

Pokemon classes

`Class Pokemon`

`Class Electric Pokemon extends Pokemon`

`Class Pikachu extends ElectricPokemon`

```
$pokemon = new Pokemon('Pikachu', 'Electric');
$pokemon->attack(); //returns a generic attack

$pikachu = new Pikachu();
$pikachu->attack(); // returns 'thunderbolt!'
```

Interfaces are like contracts

- all methods are public
- the class must implement the methods

Creating a Pokemon Interface

```
interface PokemonInterface {

  public function setPokemonName($name);
  public function setPokemonType($type);

  public function getPokemonName();
  public function getPokemonType();

  public function attack();
  public function getStrength();
  public function getWeakness();

}
```

Implementing a Pokemon Interface

```
interface PokemonInterface {
  public function getPokemonType();
  /* more methods */
}
```

Implementing a Pokemon Interface

```php
interface PokemonInterface {
  public function getPokemonType();
  /* more methods */
}
```

```php
class ElectricPokemon implements PokemonInterface {
  $type = 'Electric';
  /* other properties and methods */

  public function getPokemonType() {
    return $this->type;
  }
}
```

Implementing a Pokemon Interface

```php
interface PokemonInterface {
  public function getPokemonType();
  /* more methods */
}
```

```php
class ElectricPokemon implements PokemonInterface {
  $type = 'Electric';
  /* other properties and methods */

  public function getPokemonType() {
    return $this->type;
  }
}
```

```php
class Pikachu extends ElectricPokemon {
  /* other properties and methods */
  public function getPokemonType() {
    return $this->type;
  }
```

Abstract classes are like

blah

blah

Abstract ElectricPokemon Class

```
interface PokemonInterface {
  public function getPokemonType();
}
```

Abstract ElectricPokemon Class

```
interface PokemonInterface {
  public function getPokemonType();
}
```

```
abstract class ElectricPokemon implements PokemonInterface {
  public function getPokemonType() {
    return 'Electric';
  }
}
```

Abstract ElectricPokemon Class

```
interface PokemonInterface {
  public function getPokemonType();
}
```

```
abstract class ElectricPokemon implements PokemonInterface {
  public function getPokemonType() {
    return 'Electric';
  }
}
```

```
class Pikachu extends ElectricPokemon { ... }
```

Abstract ElectricPokemon Class

```
interface PokemonInterface {
  public function getPokemonType();
}
```

```
abstract class ElectricPokemon implements PokemonInterface {
  public function getPokemonType() {
    return 'Electric';
  }
}
```

```
class Pikachu extends ElectricPokemon { ... }
```

```
$pikachu = new Pikachu();
$pikachu->getPokemonType(); // returns 'Electric'
```

Interfaces in Drupal

```
interface FormInterface {
  public function getFormId();
}
```

Interfaces in Drupal

```
interface FormInterface {
  public function getFormId();
}
```

```
abstract class FormBase implements FormInterface { }
```

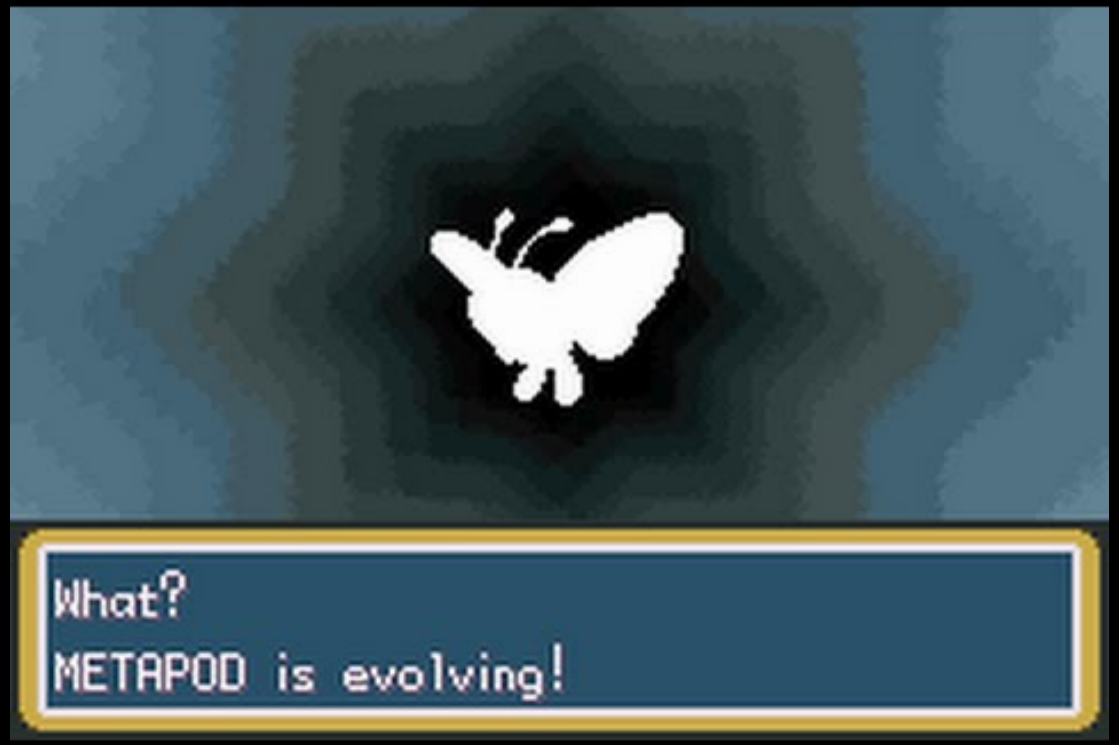
Interfaces in Drupal

```
interface FormInterface {
  public function getFormId();
}
```

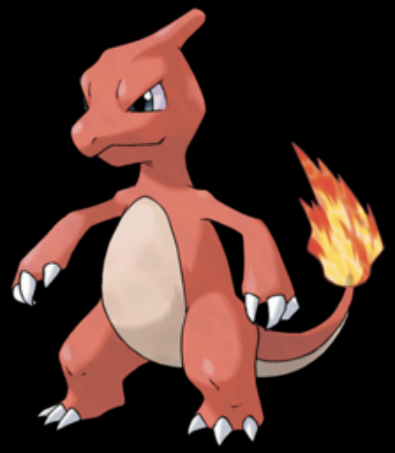
```
abstract class FormBase implements FormInterface { }
```

```
class UserLoginForm extends FormBase {
  public function getFormId() {
    return 'user_login_form';
  }
}
```

Your Pokemon is evolving!



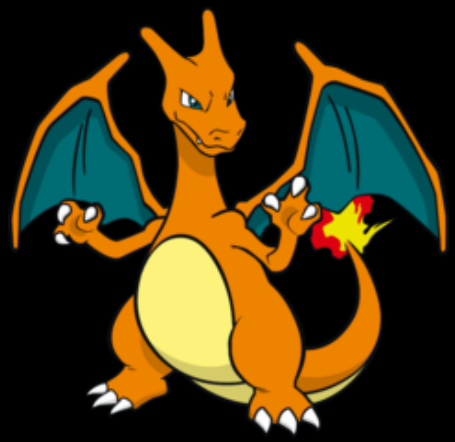
Our Pokemon is Evolving!



Charmeleon

type: Fire

attack:



Charizard

type: fire

attack:

Traits are like code snippets

- code reuse
- avoid inheritance
- cannot be instantiated

Creating a Pokemon Evolution Trait

```
trait PokeEvolutionTrait {
  public function evolve() {
    return new $this->nextStage;
  }
}
```

Creating a Pokemon Evolution Trait

```
trait PokeEvolutionTrait {
  public function evolve() {
    return new $this->nextStage;
  }
}
```

```
class Charmeleon extends FirePokemon {
  use PokeEvolutionTrait;
  public $nextStage = 'Charizard';
}
```

Creating a Pokemon Evolution Trait

```
trait PokeEvolutionTrait {
  public function evolve() {
    return new $this->nextStage;
  }
}
```

```
class Charmeleon extends FirePokemon {
  use PokeEvolutionTrait;
  public $nextStage = 'Charizard';
}
```

```
class Charizard extends FirePokemon { ... }
```

Creating a Pokemon Evolution Trait

```php
trait PokeEvolutionTrait {
  public function evolve() {
    return new $this->nextStage;
  }
}
```

```php
class Charmeleon extends FirePokemon {
  use PokeEvolutionTrait;
  public $nextStage = 'Charizard';
}
```

```php
class Charizard extends FirePokemon { ... }
```

```php
$charmeleon = new Charmeleon();
$charmeleon->evolve(); // Returns a Charizard object!
```

Traits in Drupal

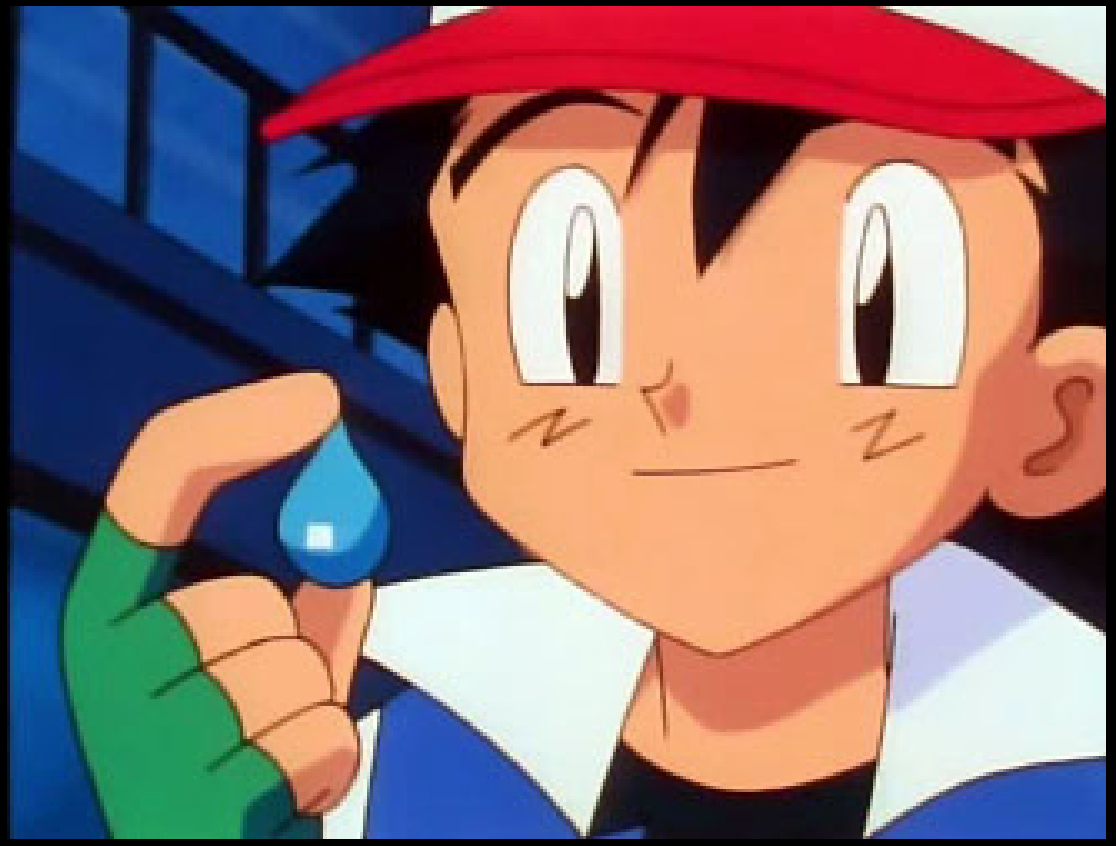
```
trait StringTranslationTrait {
  protected function t(...) {
    return new TranslatableMarkup($string, $args,
     $options, $this->getStringTranslation());
  }
}
```

Traits in Drupal

```
trait StringTranslationTrait {
  protected function t(...) {
    return new TranslatableMarkup($string, $args,
     $options, $this->getStringTranslation());
  }
}
```

```
abstract class PluginBase extends ComponentPluginBase {
  use StringTranslationTrait;
}
```

You won your first badge!



plugins are like functional lego blocks

many types of plugins

different behaviors | common interface

Pokemon Profile Block

```
/**
 * Provides a 'PokemonProfileBlock' block.
 *
 * @Block(
 *  id = "pokemon_profile_block",
 *  admin_label = @Translation("Profile Block for a Pokemon"),
 * )
 */
class PokemonProfileBlock extends BlockBase implements ContainerFactoryPluginInterface { ..
```

Annotations-based plugins

annotation data lives in the same file

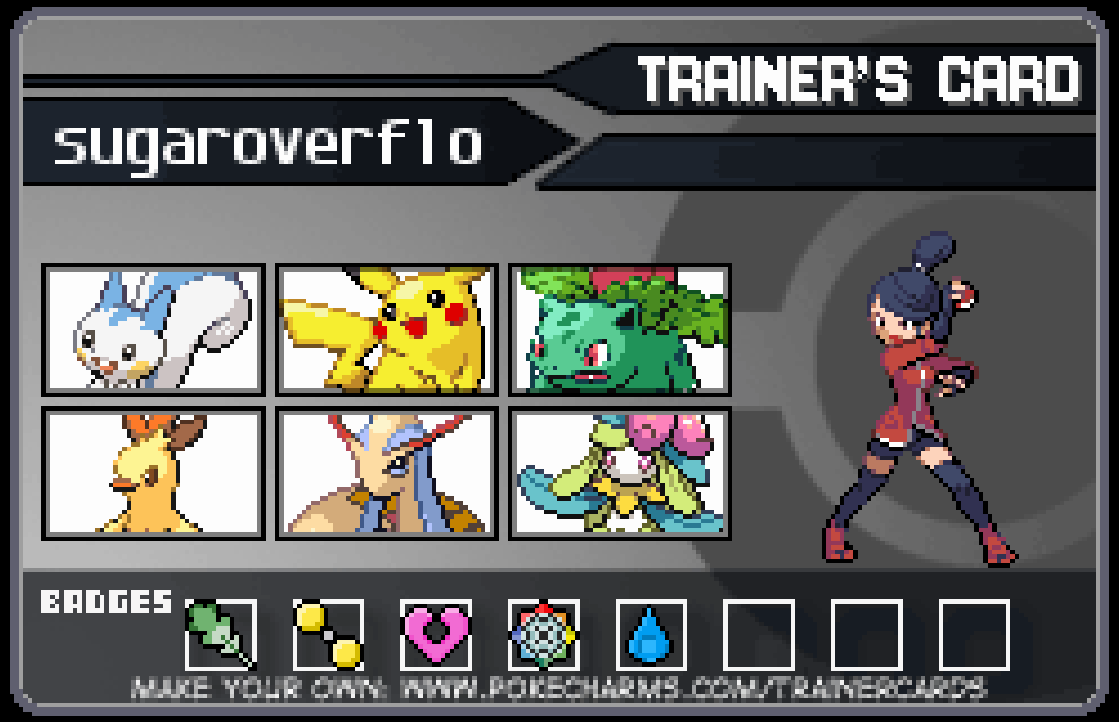
allows for complex nested data

Pokemon Profile Block

```
/**
 * (annotation here)
 */
class PokemonProfileBlock extends BlockBase {

  public function build() {
    $render = [];

    // build $stuff of pokemon info.

    $render['pokemon_container'] = [
      '#type' => 'container',
      '#markup' => $stuff,
    ];

    return $render;
  }
}
```

Pokemon Leagues

To be the very best



services are like swappable operations

- same function | swappable code
- globally available
- usually an interface defining methods

Creating a Pokemon Service

```
interface PokeDataInterface {
   public function getYearlyStats(PokemonInterface $pokemon, int $year);
}
```

Creating a Pokemon Service

```
interface PokeDataInterface {
   public function getYearlyStats(PokemonInterface $pokemon, int $year);
}
```

```
class PokeDataService implements PokeDataInterface {

  public function getYearlyStats(PokemonInterface $pokemon, int $year) {
    // does query stuff
    return $array_of_data;
  }
}
```

Calling our Pokemon Service

```
$pikachu = new Pikachu();
```

Calling our Pokemon Service

```
$pikachu = new Pikachu();
```

```
$pokeDataHelper= \Drupal::service('pokemon.pokedataservice');
```

Calling our Pokemon Service

```
$pikachu = new Pikachu();
```

```
$pokeDataHelper= \Drupal::service('pokemon.pokedataservice');
```

```
$data = $pokeDataHelper->getYearlyStats($pikachu, '2018');
```

Services in Drupal

Drupal core provides a lot of services

`core.services.yml`

```
  current_user:
    class: Drupal\Core\Session\AccountProxy
```

Services in Drupal

Drupal core provides a lot of services

`core.services.yml`

```
  current_user:
    class: Drupal\Core\Session\AccountProxy
```

```
$current_user = \Drupal::service('current_user');
// access something from the $current_user
```

Services in Drupal are swappable

```
  email.validator:
    class: Egulias\EmailValidator\EmailValidator
```

```
class ContactFormEditForm extends EntityForm {

  if ($this->emailValidator->isValid($recipient)) {
      //do stuff
  }

}
```

When possible, inject your services

- pass as arguments to a constructor
- or use setter methods

When possible, inject your services

- pass as arguments to a constructor
- or use setter methods

```
$pokeDataService = \Drupal::service('pokemon.pokedataservice');
```

Injecting the pokeDataService I

```
/**
 * (annotation here)
 */
class PokemonProfileBlock extends BlockBase implements ContainerFactoryPluginInterface {
  // do stuff
}
```

Injecting the pokeDataService II

```
/**
 * (annotation here)
 */
class PokemonProfileBlock extends BlockBase implements ContainerFactoryPluginInterface {
  protected $pokeDataService;

  public function __construct( PokeDataInterface $pokeDataService) {
    $this->pokeDataService = $pokeDataService;
  }
}
```

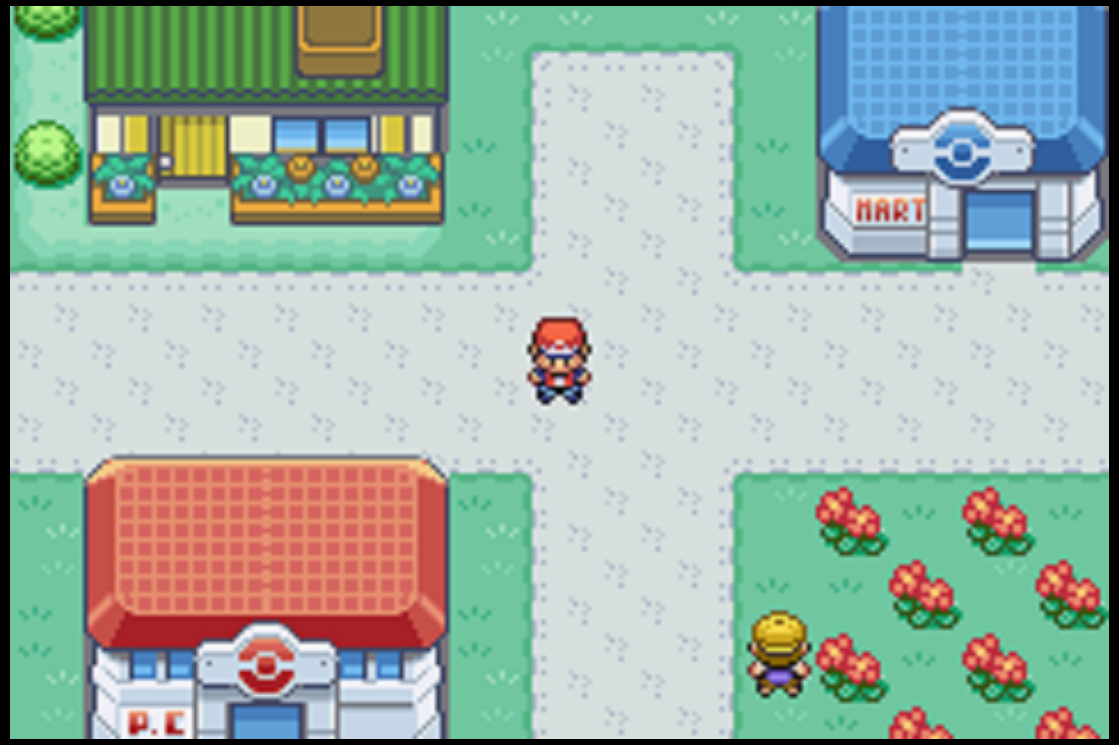
Injecting the pokeDataService III

```
/**
 * (annotation here)
 */
class PokemonProfileBlock extends BlockBase implements ContainerFactoryPluginInterface {
  protected $pokeDataService;

  public function __construct( PokeDataInterface $pokeDataService) {
    $this->pokeDataService = $pokeDataService;
  }

  public static function create(ContainerInterface $container) {
    return new static(
      $container->get('pokemon.pokedataservice'),
    );
  }
}
```

Injecting the pokeDataService IV

```
/* (annotation here) */
class PokemonProfileBlock extends BlockBase implements ContainerFactoryPluginInterface {
  protected $pokeDataService;

  public function __construct( PokeDataInterface $pokeDataService) {
    $this->pokeDataService = $pokeDataService;
  }

  public static function create(ContainerInterface $container) {
    return new static(
      $container->get('pokemon.pokedataservice'),
    );
  }

  public function build() {
    $pokeDataService->getYearlyStats($this->pokemon, '2018');
    // do block stuff
  }
}
```

The Adventure Continues



Thank you!

slides: bit.ly/slideslink

feedback:

special thank you to @cottser