Objects

- Can contain multiple variables or properties, similar to how an associative array can contain multiple elements, but with potentially limited scope
- Can have associated functions or methods that act on those properties
- Both are referenced using the single arrow operator

```php
$object = new stdClass;
$object->meaning_of_life = 42;

$pdo = new PDO('...');
$result = $pdo->query('...');
```

Classes

- "Blueprint" for creating objects referred to as instances, a process called instantiation
- Collection of property and method definitions
- Provides a level of scope shared across multiple methods and accessed using the special `$this` variable

Classes

```php
class Order {
  private $items = [];
  private $tax = 0;
  private $discount = 0;
  public function addItem($id, $price, $quantity) {
    $this->items[] = [
      'id' => $id,
      'price' => $price,
      'quantity' => $quantity, ];
  }
  public function setTax($tax) {
    $this->tax = $tax;
  }
  public function setDiscount($discount) {
    $this->discount = $discount;
  }
  public function getTotal() {
    // ...
  }
}
```

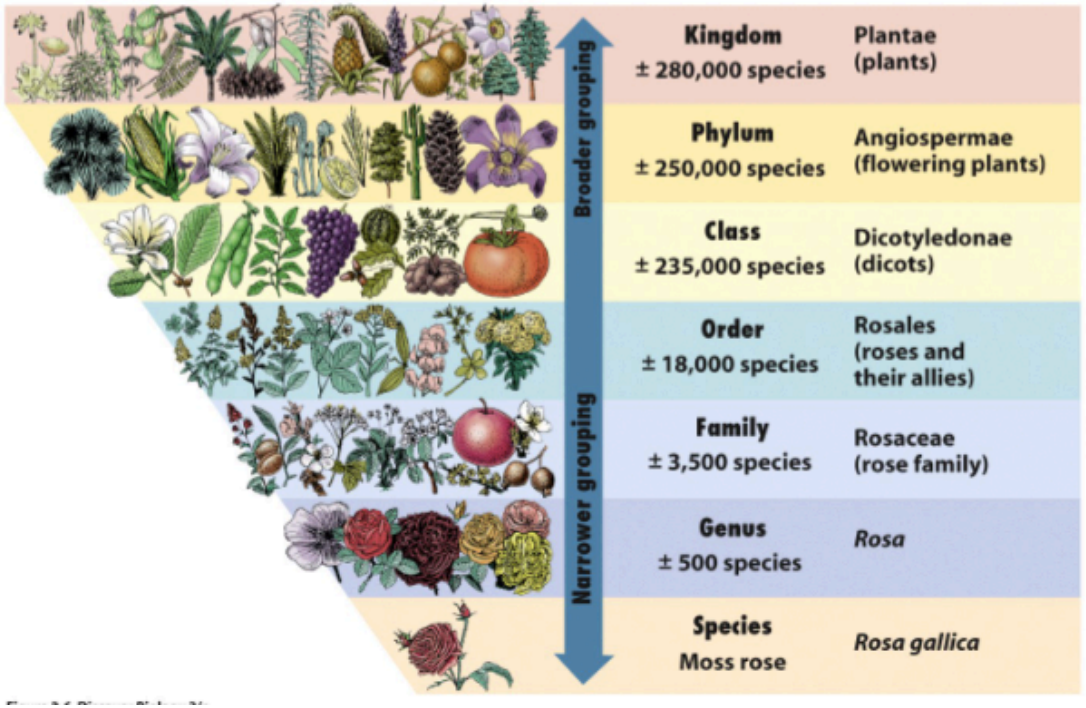
Instantiation

```php
$order1 = new Order;
$order1->addItem('phppro1', 39.95, 1);
$order1->setTax(0.075);
$order1->setDiscount(0.1);

$order2 = new Order;
$order2->addItem('web-scraping', 39.99, 1);
$order2->setTax(0);
$order2->setDiscount(0);
```

Inheritance

"Inheritance is a way to establish is-a relationships between objects." ~ Wikipedia



Subclasses

- A subclass inherits the code defined in another class, a superclass, plus whatever that subclass contains
- The `extends` keyword is used to specify the superclass of a subclass
- Subclasses can override superclass method implementations
- `parent::` can be used in overrides to call the superclass implementation of the same method

Subclasses

```php
class ShippedOrder extends Order {
    protected $shipping = 0;
    public function setShipping($shipping) {
        $this->shipping = $shipping;
    }
    public function getTotal() {
        return parent::getTotal() + $this->shipping;
    }
}
```

Visibility

- `public` allows access outside and inside the class
- `protected` allows access inside the class and its subclasses
- `private` allows access inside the class
- `const` allows read-only access independent of individual instances

Public Access Modifier

```php
class Order {
    public $tax = 0.08;
    public function getTotal() {
        // ...
    }
}
$order = new Order;
var_dump($order->tax); // float(0.08)
var_dump($order->getTotal()); // float(1.08)
$order->tax = 0.12;
var_dump($order->tax); // float(0.12)
var_dump($order->getTotal()); // float(1.12)
// ^ Work the same for subclasses
```

Protected Access Modifier

```php
class Order {
    protected $tax;
    protected function getSubtotal() {
        // ...
    }
    public function getTotal() {
        return $this->getSubtotal() + $this->tax;
    }
}

class ShippedOrder extends Order {
    protected $shipping;
    public function getTotal() {
        return $this->getSubtotal() + $this->tax + $this->shipping;
    }
}
```

Protected Access Modifier

```php
$order = new ShippedOrder;

var_dump($order->tax);
// PHP Fatal error: Cannot access protected property
// ShippedOrder::$tax

var_dump($order->getSubtotal());
// PHP Fatal error: Call to protected method
// ShippedOrder::getSubtotal()

var_dump($order->getTotal());
// float(1.08)
```

Private Access Modifier

```php
class Order {
    private $tax = 0.08;
    private function getSubtotal() {
        // ...
    }
    public function getTotal() {
        return $this->getSubtotal() + $this->tax;
    }
}
class ShippedOrder extends Order {
    public function getTax() {
        return $this->tax;
    }
    public function getTotal() {
        return $this->getSubtotal() + $this->tax + $this->shipping;
    }
}
```

Constants

```php
class Order {
    const TAX = 0.08;
    public function getTax() {
        return self::TAX;
    }
}
class ShippedOrder extends Order {
    // ...
}
```

Abstraction

"Abstraction is the process by which data and programs are defined with a representation similar in form to its meaning (semantics), while hiding away the implementation details... a concept or idea not associated with any specific instance." ~ Wikipedia

Abstraction

- Abstract classes cannot be instantiated, only extended
- Abstract methods, designated using the `abstract` modifier, have no implementation in the superclass that declares them; subclasses must implement them
- Abstract classes can have both abstract and non-abstract methods

Abstract Classes

```php
abstract class Order {
    private $tax = 0.08;
    protected function getTax() {
        return $this->tax;
    }
    protected function getSubtotal() {
        // ...
    }
    abstract public function getTotal();
}
class ShippedOrder extends Order {
    protected $shipping = 1;
    public function getTotal() {
        return $this->getSubtotal() + $this->getTax() + $this->shipping;
    }
}
```

Abstract Classes

```php
$order = new Order;
// PHP Fatal error: Cannot instantiate abstract class Order

$order = new ShippedOrder;
var_dump($order->getTotal()); // float(2.08)

class DigitalOrder extends Order { }
// PHP Fatal error: Class DigitalOrder contains 1 abstract
// method and must therefore be declared abstract or
// implement the remaining methods (Order::getTotal)
```

Polymorphism

"The primary usage of polymorphism... is the ability of objects belonging to different types to respond to method... or property calls of the same name, each one according to an appropriate type-specific behavior."~ Wikipedia

Class Typehints

- Class names can precede method parameter names to require that those parameters be instances of those classes or their subclasses at runtime
- Violations of this requirement result in a catchable fatal error

Class Typehints

```
1  class Dog { }
2  class Dachshund extends Dog { }
3  class DogWasher {
4      public function wash(Dog $dog) { }
5  }
6  class Cat { }
7
8  $dog = new Dog;
9  $dachshund = new Dachshund;
10 $dogwasher = new DogWasher;
11 $cat = new Cat;
```

Class Typehints

```
1  // $dog matches Dog
2  $dogwasher->wash($dog);
3
4  // $dachshund matches Dachshund, a subclass of Dog
5  $dogwasher->wash($dachshund);
6
7  // $cat doesn't match Dog or Dachshund
8  $dogwasher->wash($cat);
9  // PHP Catchable fatal error: Argument 1 passed to
10 // DogWasher::wash() must be an instance of Dog,
11 // instance of Cat given
```

Interfaces

- Interface methods have no implementation; a class that implements an interface must implement its methods
- Classes can implement multiple interfaces, whereas they can only extend a single superclass
- Interface names can precede method parameter names to require that those parameters be instances of classes that implement those interfaces

Interfaces

```php
interface Configurable {
    public function setConfig(array $config);
}

interface Outputable {
    public function output();
}
```

Interfaces

```php
class Foo implements Configurable, Outputable {
    protected $config = [];
    public function setConfig(array $config) {
        $this->config = $config;
    }
    public function output() {
        var_dump($this);
    }
}
```

Interfaces

```
1  class Configurator {
2      public function configurate(Configurable $configurable) {
3          // ...
4      }
5  }
6  $foo = new Foo;
7  $configurator = new Configurator;
8  $configurator->configurate($foo);
9  // works because $foo is an instance of Foo which
10 // implements Configurable
```

Interface Typehints

```php
interface Washable { }
class Washer {
    public function wash(Washable $washable) { }
}
class Dog implements Washable { }
class Cat { }

$washer = new Washer;
$dog = new Dog;
$cat = new Cat;
```

Interface Typehints

```php
// $dog matches Dog, which implements Washable
$washer->wash($dog);

// $cat matches Cat, which does not implement Washable
$washer->wash($cat);
// Catchable fatal error: Argument 1 passed to Washer::wash()
// must implement interface Washable, instance of Cat given
```

Composition

"Composition over inheritance... is a technique by which classes may achieve polymorphic behavior and code reuse by containing [instances of] other classes that implement the desired functionality instead of through inheritance." ~ Wikipedia

Constructor

"... a constructor in a class is a special type of subroutine … [that] prepares the new object for use, often accepting arguments that the constructor uses to set required member variables." ~ Wikipedia

Destructor

"... a destructor... is a method which is automatically invoked when the object is destroyed... Its main purpose is to free the resources... which were acquired by the object along its life cycle and/or deregister from other entities which may keep references to it." ~ Wikipedia

Constructors / Destructors

```php
class Foo {
    protected $config;
    public function __construct(array $config) {
        $this->config = $config;
    }
    public function getConfig() {
        return $this->config;
    }
    public function __destruct() {
        // ...
    }
}
```

Constructors / Destructors

```php
$foo = new Foo(['bar' => 'baz']);
// Invokes __construct()

var_dump($foo->getConfig());
// Output: array(1) { ["bar"]=> string(3) "baz" }

unset($foo);
// Invokes __destruct()
```