# Object-oriented PHP: Classical Inheritance Model

## INHERITANCE

**Daryl K Wood**
CTO DATASHUTTLE.NET

@datashuttle | www.datashuttle.net | www.linkedin.com/in/datashuttle

# Course Overview

Introduction

Knowledge prerequisite

Software prerequisite

Goals for this course

# Knowledge Attainment

Understand the PHP inheritance model

Recognize inheritance architecture

Efficient coding

# Course Topics

- Class inheritance
- Class member visibilities
- Property and method overrides
- MVC design concepts
- Software entities as objects
- Object aspect design
- PHP traits and class relationships
- Abstract classes and interfaces

# Objects Represent Something

Car

Train

Airplane

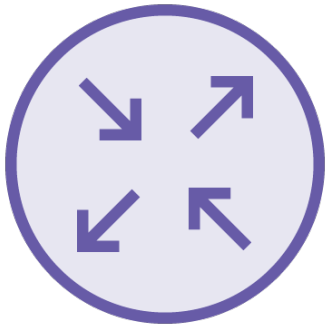
Doctor

Hospital

Route or Destination

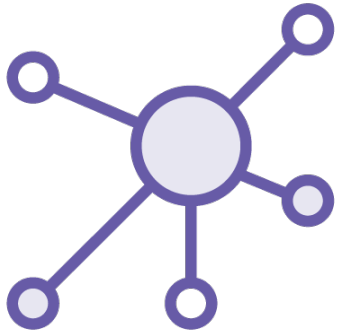
# Objects Represent Something in Software

Service



Controller

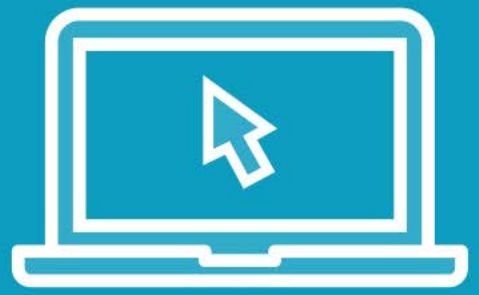
Database



Connection

# Confused?

# Demo



Default properties and methods

Represented as individual objects

Can contain unique members

Can contain duplicate members between objects

# PHP Class Inheritance

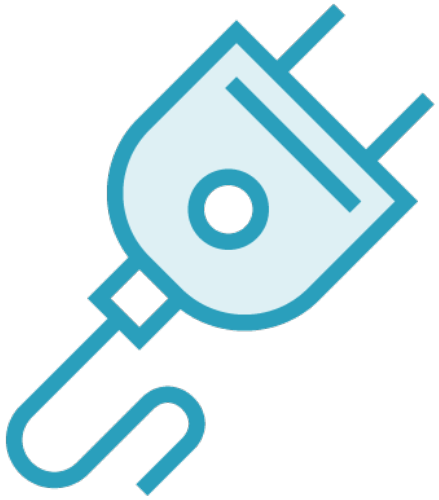
You've Got the **Power!**

**Efficient** Architectures

**No More** Duplication

Vertical **Inheritance Model**

# Inheritance Puts the Power in OOP



Cornerstone of OOP

Part magic, part elegance

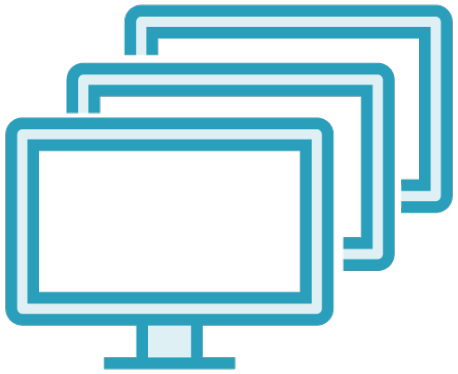
Simple or complex

# Efficient Architectures

**Code organization**

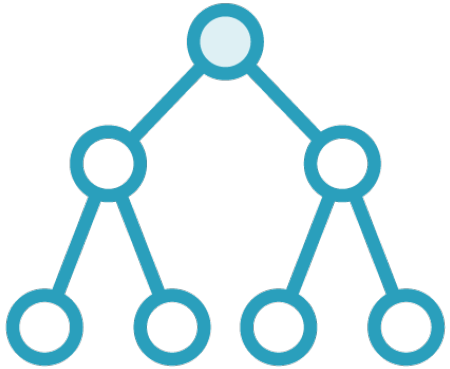
**Requires forethought**

# No More Duplication



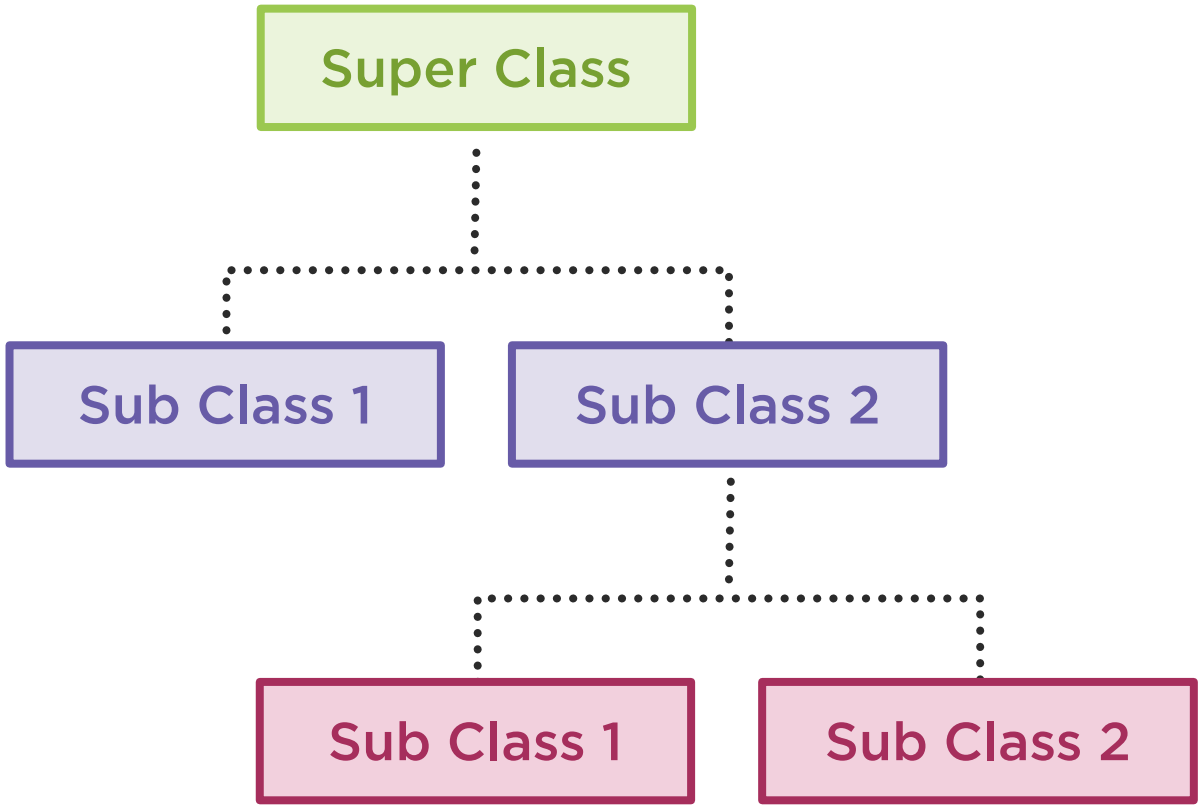
- Eliminate duplication
- Reduce overhead
- Reduce spaghetti code

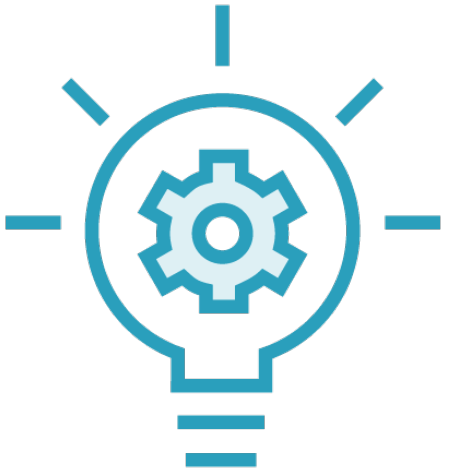
# A Vertical Inheritance Model



**Vertically cascading**

**Seek commonality**

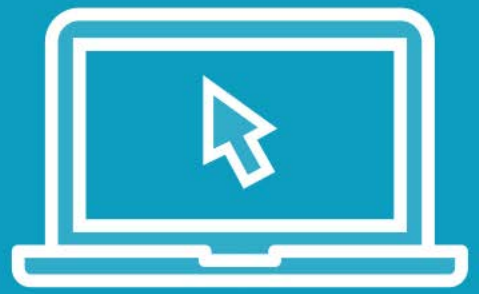

Super Class
Sub Class 1
Sub Class 2
Sub Class 1
Sub Class 2



# Expanding the Classes

- Add base and subclass functionality
- Control with the protected visibility
- Using getters and setters
- Implement chaining

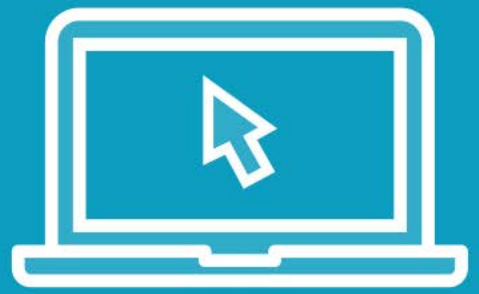
# Demo



Refactoring a base class

Refactoring sub classes

# Demo



Control of application flow

What can and cannot be changed

From where

# `get*() | set*()`

Used as accessors/mutators for inaccessible properties.

# Expanding the Classes

Returning `$this`

**Chaining** Methods

Step through the
**changes to date**

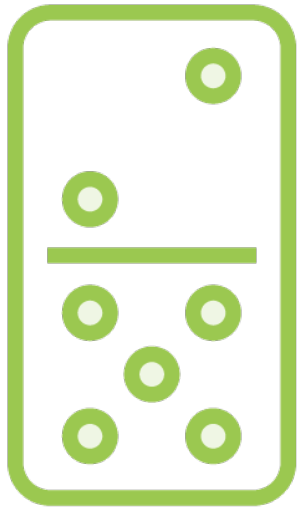
**Next** Steps

# Returning `$this`

**Return the instance `$this`**

**No temporary variables required**

# Chaining Methods



What is method chaining?

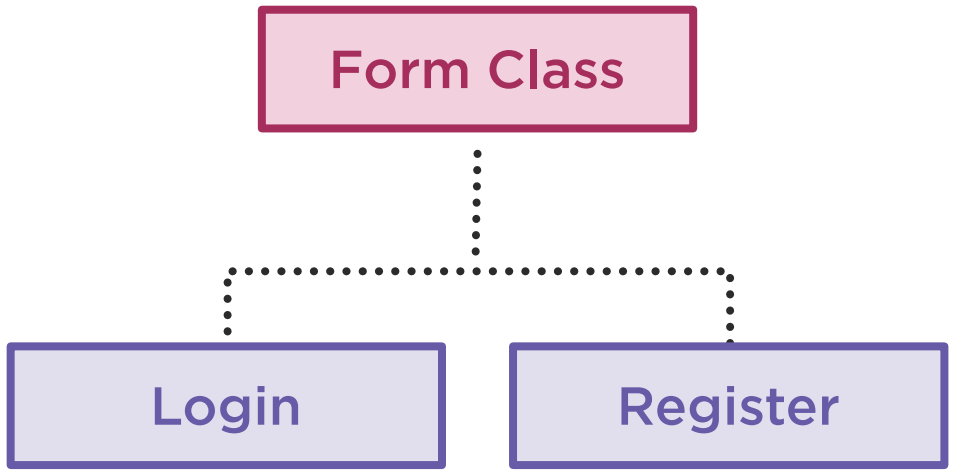
Why chain?

A train of action or domino effect

# On to the Form Application

**Implement inheritance in the form application**

# Applying Inheritance to the Form Application




Refactoring with inheritance

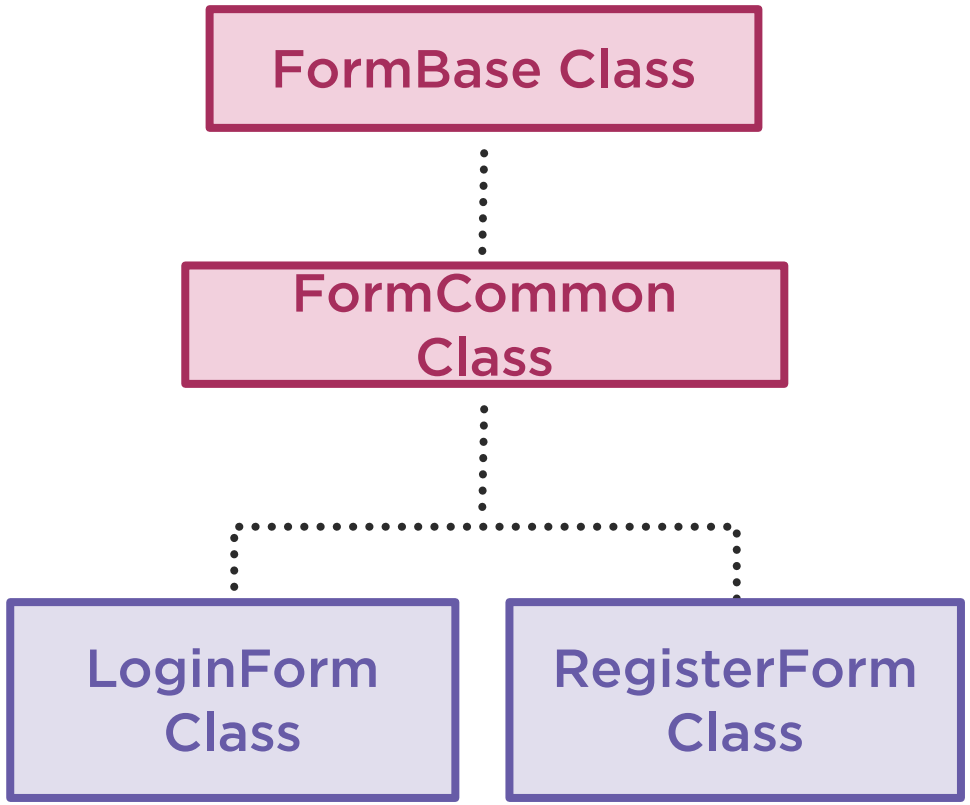
Create a form base class

A register form sub class

A login form sub class

Form application review

# Inheritance in the Form Classes



# Inheritance in the Form Classes

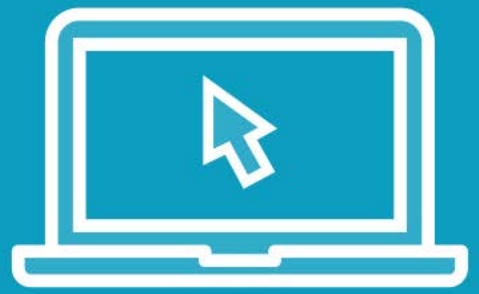
Refactoring to
a **form base** class

Create a form
**common** class

Create a **registration**
form sub class

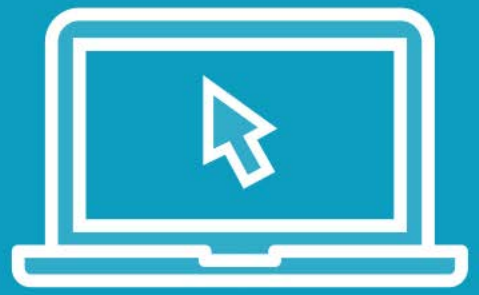
Create a **Login**
form sub class

# Demo



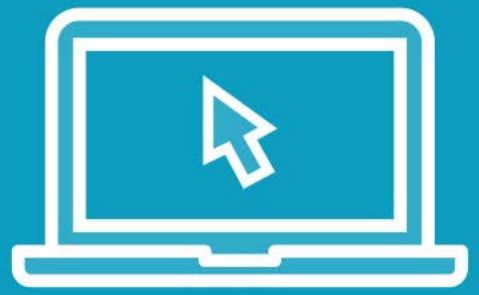
## Refactoring a form base class

# Demo



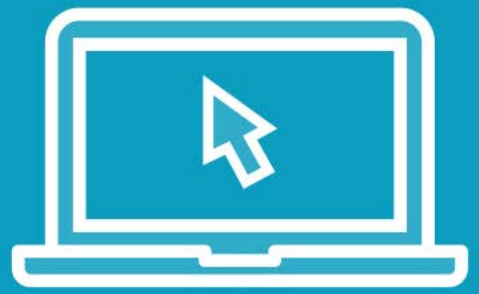
## Create a form common class

# Demo



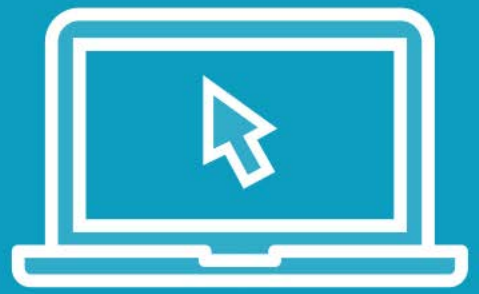
## Create a registration form sub class

# Demo



## Create a login form sub class

# Demo

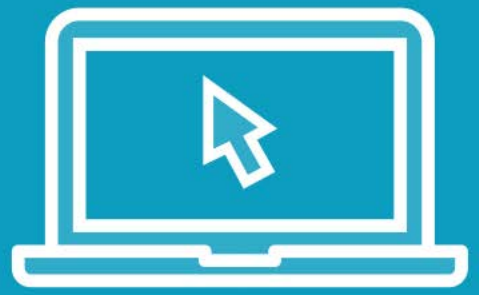


Run the code

Re-factored forms

Command and base classes

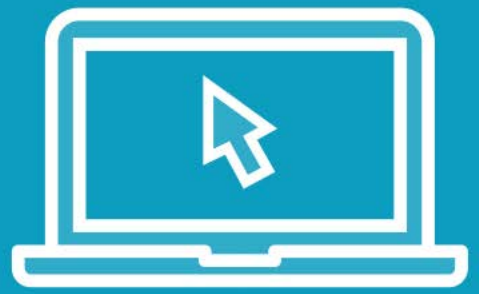
# Demo



Refactoring to an input base class

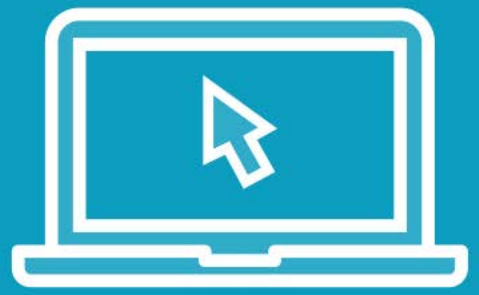
Create inheritance with input classes

# Demo



## Auto loading classes

# Demo



Standard PHP Library (SPL)

SPL auto load

# Module Summary

Super and sub classes

How to inherit and why

Implement in form and input classes

Autoloading