MVC (Model, View, Controller)

Disclaimer: вы смотрите просто запись лекции, это **НЕ** специально подготовленный видеокурс!

Intro

MVC (Model, View, Controller) is a design pattern used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display.

Some other design patterns are based on MVC, such as MVVM (Model-View-View-Model), MVP (Model-View-Presenter), and MVW (Model-View-Whatever).

Disclaimer

Although MVC is rather popular in web-applications development, many people reasonably say it works good for UI part only, but the application "as a whole" may (and should?) be built on more resource-efficient principles.

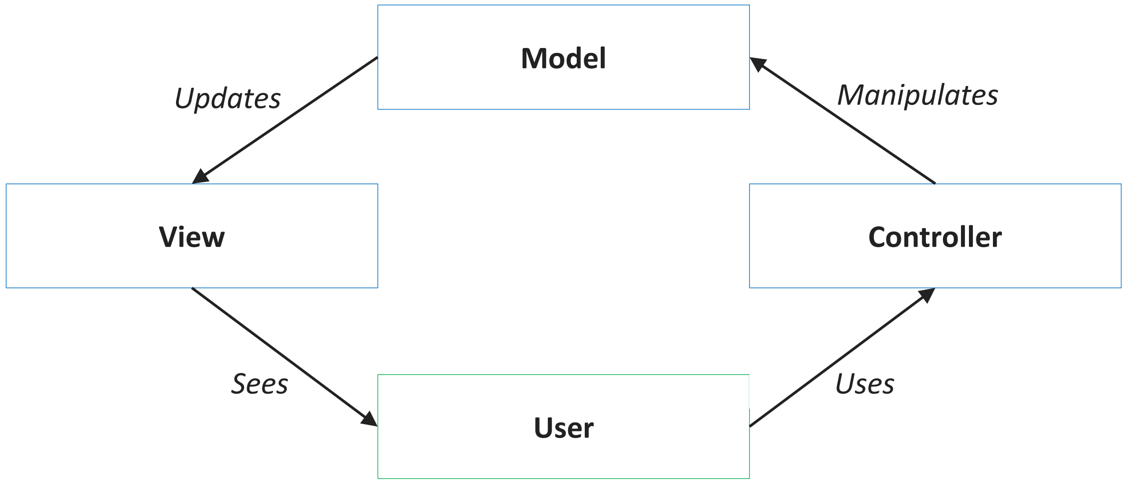
A lot of useful information

Please refer to these sources for details:
http://www.sitepoint.com/the-mvc-pattern-and-php-1/
http://www.sitepoint.com/the-mvc-pattern-and-php-2/

P.S. Don’t be confused that these articles are written in 2013th. The MVC concept itself was created in 1970s.

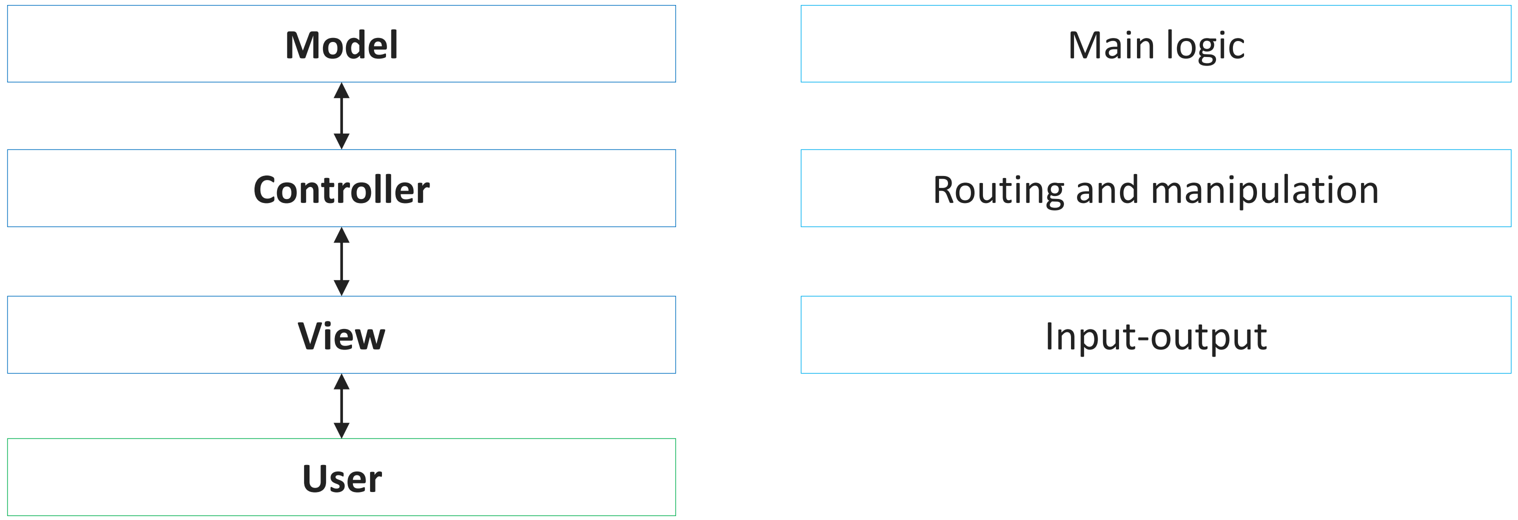
A deeper look…

Model manages data and business logic.
View handles layout and display.
Controller routes commands to the model and view parts.



A deeper look, closer to reality

Model manages data and business logic.
View handles layout and display.
Controller routes commands to the model and view parts.



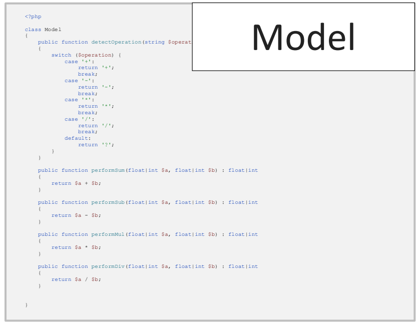
Sample

As the code is rather huge, let’s review it in the IDE.

P.S. But just in case you don’t have the handouts, here are copiable samples (see the next two slides).

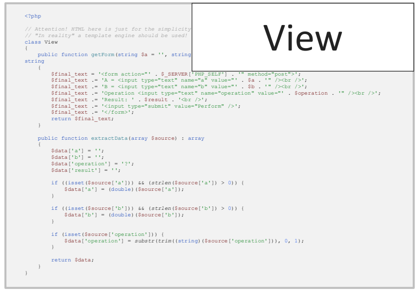
Simplified approach

In this approach some logic are placed in so-called “entry point” to simplify the overall logic. (See “01_simplified” in handouts.)



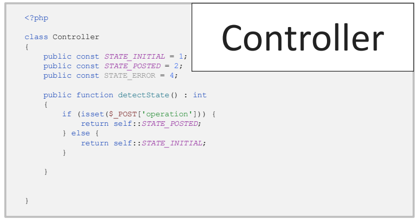
Model

```php
<?php

class Model
{
    public function detectOperation(string $operat
    {
        switch ($operation) {
            case '+':
                return '+';
                break;
            case '-':
                return '-';
                break;
            case '*':
                return '*';
                break;
            case '/':
                return '/';
                break;
            default:
                return '?';
        }
    }

    public function performSum(float|int $a, float|int $b) : float|int
    {
        return $a + $b;
    }

    public function performSub(float|int $a, float|int $b) : float|int
    {
        return $a - $b;
    }

    public function performMul(float|int $a, float|int $b) : float|int
    {
        return $a * $b;
    }

    public function performDiv(float|int $a, float|int $b) : float|int
    {
        return $a / $b;
    }


}
```



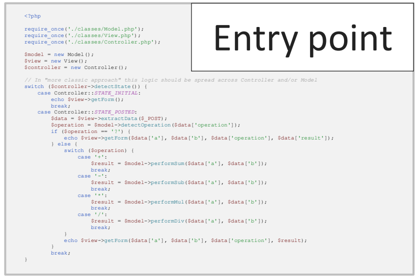
View

```php
<?php

// Attention! HTML here is just for the simplicity
// "In reality" a template engine should be used!
class View
{
    public function getForm(string $a = '', string
string
    {
        $final_text = '<form action="' . $_SERVER['PHP_SELF'] . '" method="post">';
        $final_text .= 'A = <input type="text" name="a" value="' . $a . '" /><br />';
        $final_text .= 'B = <input type="text" name="b" value="' . $b . '" /><br />';
        $final_text .= 'Operation <input type="text" name="operation" value="' . $operation . '" /><br />';
        $final_text .= 'Result: ' . $result . '<br />';
        $final_text .= '<input type="submit" value="Perform" />';
        $final_text .= '</form>';
        return $final_text;
    }

    public function extractData(array $source) : array
    {
        $data['a'] = '';
        $data['b'] = '';
        $data['operation'] = '?';
        $data['result'] = '';

        if ((isset($source['a'])) && (strlen($source['a']) > 0)) {
            $data['a'] = (double)($source['a']);
        }

        if ((isset($source['b'])) && (strlen($source['b']) > 0)) {
            $data['b'] = (double)($source['b']);
        }

        if (isset($source['operation'])) {
            $data['operation'] = substr(trim((string)($source['operation'])), 0, 1);
        }

        return $data;
    }
}
```



Controller

```php
<?php

class Controller
{
    public const STATE_INITIAL = 1;
    public const STATE_POSTED = 2;
    public const STATE_ERROR = 4;

    public function detectState() : int
    {
        if (isset($_POST['operation'])) {
            return self::STATE_POSTED;
        } else {
            return self::STATE_INITIAL;
        }

    }


}
```

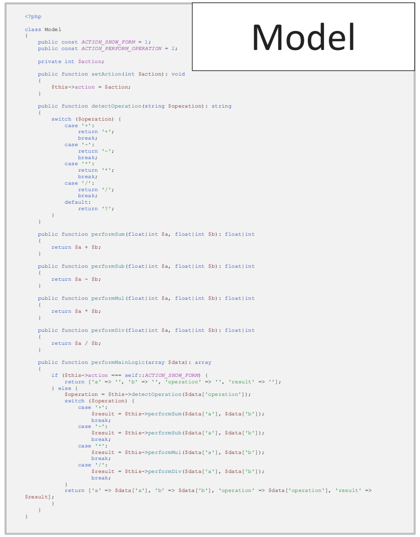


Entry point

```php
<?php

require_once('./classes/Model.php');
require_once('./classes/View.php');
require_once('./classes/Controller.php');

$model = new Model();
$view = new View();
$controller = new Controller();

// In "more classic approach" this logic should be spread across Controller and/or Model
switch ($controller->detectState()) {
    case Controller::STATE_INITIAL:
        echo $view->getForm();
        break;
    case Controller::STATE_POSTED:
        $data = $view->extractData($_POST);
        $operation = $model->detectOperation($data['operation']);
        if ($operation == '?') {
            echo $view->getForm($data['a'], $data['b'], $data['operation'], $data['result']);
        } else {
            switch ($operation) {
                case '+':
                    $result = $model->performSum($data['a'], $data['b']);
                    break;
                case '-':
                    $result = $model->performSub($data['a'], $data['b']);
                    break;
                case '*':
                    $result = $model->performMul($data['a'], $data['b']);
                    break;
                case '/':
                    $result = $model->performDiv($data['a'], $data['b']);
                    break;
            }
            echo $view->getForm($data['a'], $data['b'], $data['operation'], $result);
        }
        break;
}
```

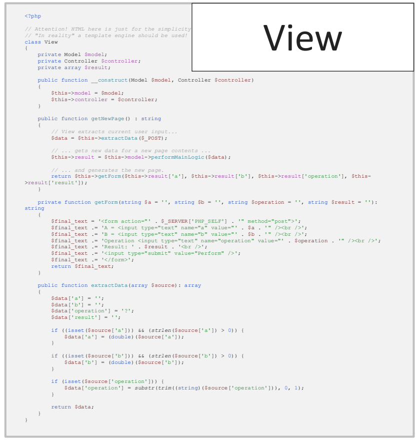
Simplified approach

This is “by-the-book” approach with all actions placed in corresponding classes and files. (See “02_classic” in handouts.)



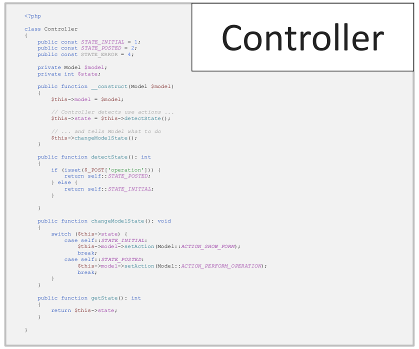
Model

```php
<?php

class Model
{
    public const ACTION_SHOW_FORM = 1;
    public const ACTION_PERFORM_OPERATION = 2;

    private int $action;

    public function setAction(int $action): void
    {
        $this->action = $action;
    }

    public function detectOperation(string $operation): string
    {
        switch ($operation) {
            case '+':
                return '+';
                break;
            case '-':
                return '-';
                break;
            case '*':
                return '*';
                break;
            case '/':
                return '/';
                break;
            default:
                return '?';
        }
    }

    public function performSum(float|int $a, float|int $b): float|int
    {
        return $a + $b;
    }

    public function performSub(float|int $a, float|int $b): float|int
    {
        return $a - $b;
    }

    public function performMul(float|int $a, float|int $b): float|int
    {
        return $a * $b;
    }

    public function performDiv(float|int $a, float|int $b): float|int
    {
        return $a / $b;
    }

    public function performMainLogic(array $data): array
    {
        if ($this->action === self::ACTION_SHOW_FORM) {
            return ['a' => '', 'b' => '', 'operation' => '', 'result' => ''];
        } else {
            $operation = $this->detectOperation($data['operation']);
            switch ($operation) {
                case '+':
                    $result = $this->performSum($data['a'], $data['b']);
                    break;
                case '-':
                    $result = $this->performSub($data['a'], $data['b']);
                    break;
                case '*':
                    $result = $this->performMul($data['a'], $data['b']);
                    break;
                case '/':
                    $result = $this->performDiv($data['a'], $data['b']);
                    break;
            }
            return ['a' => $data['a'], 'b' => $data['b'], 'operation' => $data['operation'], 'result' =>
$result];
        }
    }
}
```



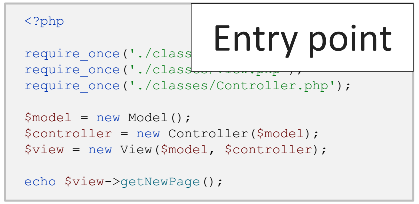
View

```php
<?php

// Attention! HTML here is just for the simplicity
// "In reality" a template engine should be used!
class View
{
    private Model $model;
    private Controller $controller;
    private array $result;

    public function __construct(Model $model, Controller $controller)
    {
        $this->model = $model;
        $this->controller = $controller;
    }

    public function getNewPage() : string
    {
        // View extracts current user input...
        $data = $this->extractData($_POST);

        // ... gets new data for a new page contents ...
        $this->result = $this->model->performMainLogic($data);

        // ... and generates the new page.
        return $this->getForm($this->result['a'], $this->result['b'], $this->result['operation'], $this-
>result['result']);
    }

    private function getForm(string $a = '', string $b = '', string $operation = '', string $result = ''):
string
    {
        $final_text = '<form action="' . $_SERVER['PHP_SELF'] . '" method="post">';
        $final_text .= 'A = <input type="text" name="a" value="' . $a . '" /><br />';
        $final_text .= 'B = <input type="text" name="b" value="' . $b . '" /><br />';
        $final_text .= 'Operation <input type="text" name="operation" value="' . $operation . '" /><br />';
        $final_text .= 'Result: ' . $result . '<br />';
        $final_text .= '<input type="submit" value="Perform" />';
        $final_text .= '</form>';
        return $final_text;
    }

    public function extractData(array $source): array
    {
        $data['a'] = '';
        $data['b'] = '';
        $data['operation'] = '?';
        $data['result'] = '';

        if ((isset($source['a'])) && (strlen($source['a']) > 0)) {
            $data['a'] = (double)($source['a']);
        }

        if ((isset($source['b'])) && (strlen($source['b']) > 0)) {
            $data['b'] = (double)($source['b']);
        }

        if (isset($source['operation'])) {
            $data['operation'] = substr(trim((string)($source['operation'])), 0, 1);
        }

        return $data;
    }
}
```



Controller

```php
<?php

class Controller
{
    public const STATE_INITIAL = 1;
    public const STATE_POSTED = 2;
    public const STATE_ERROR = 4;

    private Model $model;
    private int $state;

    public function __construct(Model $model)
    {
        $this->model = $model;

        // Controller detects use actions ...
        $this->state = $this->detectState();

        // ... and tells Model what to do
        $this->changeModelState();
    }

    public function detectState(): int
    {
        if (isset($_POST['operation'])) {
            return self::STATE_POSTED;
        } else {
            return self::STATE_INITIAL;
        }

    }

    public function changeModelState(): void
    {
        switch ($this->state) {
            case self::STATE_INITIAL:
                $this->model->setAction(Model::ACTION_SHOW_FORM);
                break;
            case self::STATE_POSTED:
                $this->model->setAction(Model::ACTION_PERFORM_OPERATION);
                break;
        }
    }

    public function getState(): int
    {
        return $this->state;
    }

}
```



Entry point

```php
<?php

require_once('./class
require_once('./classes/View.php');
require_once('./classes/Controller.php');

$model = new Model();
$controller = new Controller($model);
$view = new View($model, $controller);

echo $view->getNewPage();
```

MVC (Model, View, Controller)

Disclaimer: вы смотрите просто запись лекции, это **НЕ** специально подготовленный видеокурс!