Capacitação OOP

Lucas de Miranda

Alguns materiais foram retirados dos slides do professor Douglas Macharet das aulas de Programação Modular (DCC052).

Motivação

Software está em constante evolução

Custo de Manutenção

Problemas de Manutenção

?>

Motivação

- Programação Estruturada
  - Instruções que mudam o estado do programa
  - Programas imperativos

- Programação Orientada a Objetos
  - Dados e procedimentos encapsulados
  - Composto por diversos objetos
  - Interação/comunicação entre os objetos

?>

Motivação

- Como resolver problemas muito grandes?
  - Dividir para conquistar
  - Construí-lo a partir de partes menores

- Funções
  - Solucionam uma parte do problema
  - Dados x Manipulação
  - Abstração fraca para problemas mais complexos

Programação Orientada a Objetos

- Softwares grande e complexos
  - Aumentar a produtividade no desenvolvimento
  - Diminuir a chance de problemas
  - Facilitar a manutenção

- Programação Orientada a Objetos
  - Tem apresentado bons resultados
  - Não é uma bala de prata!

ABSTRAÇÃO

*“Lo-lo-lo-look how he, he moooooves morty *BURP*.*

This defies logic, Mo-Morty.”

Exemplo prático - Carro

Vamos tentar modelar um carro, utilizando algumas operações básicas.

?>

Exemplo prático - Carro

Vamos tentar modelar um carro, utilizando algumas operações básicas.

⇒ Acelerar
⇒ Frear
⇒ Medir a velocidade Atual

?>

Exemplo prático - Carro

Todo carro precisa de combustível para funcionar.

?>

Carro - Mãos a Obra

Quais são nossos atributos?

?>

Carro - Mãos a Obra

Quais são nossos atributos?

Velocidade Atual, Gasolina Atual, Gasolina Máxima

?>

Carro - Mãos a Obra

Quais são nossos atributos?
Velocidade Atual, Gasolina Atual, Gasolina Máxima

Quais são nossos procedimentos?

Carro - Mãos a Obra

Quais são nossos atributos?
Velocidade Atual, Gasolina Atual, Gasolina Máxima

Quais são nossos procedimentos?
Acelerar, Frear, Verificar Combustível

Carro - Mãos a Obra

Quais são nossos atributos?
Velocidade Atual, Gasolina Atual, Gasolina Máxima

Quais são nossos procedimentos?
Acelerar, Frear, Verificar Combustível
Adicionar Combustível, Remover Combustível(?)

Carro - Mãos a Obra

O que foi visto é a programação estruturada!

O conceito de abstração é usado constantemente

Facilitamos a manutenção

?>

Carro - Mãos a Obra

O que foi visto é a programação estruturada!

O conceito de abstração é usado constantemente

Facilitamos a manutenção

Como passar para Orientação a Objetos?

?>

Carro - Mãos a Obra

O que todas as funções têm em comum?

?>

Classes

- Representam uma categoria de elementos
  - O Objeto representa um item em particular
- Não existem no contexto da execução!
- Definem uma lógica estática
  - Relacionamentos entre classes não mudam
  - Relacionamentos entre objetos são dinâmicos

?>

Classes

Não se ganha nenhum poder computacional com OOP ao se comparar com programação estruturada.

A diferença consiste na manutenibilidade.

?>

Classes

```php
class Elefante
{
    private $idade;
    private $peso;

    function __construct($idade, $peso)
    {
        $this->idade = $idade;
        $this->peso = $peso;
    }
}
```

?>

Classes

```php
class Elefante
{
    /* ... */

    private function calcularRiscoDeDoenca($alfa) {
        /* ... */
        return $this->atributo * $alfa;
    }

    public function mostrarRiscoDeDoenca($alfa) {
        echo "O Risco de doença é de: " .
            $this->calcularRiscoDeDoenca($alfa);
    }
}
```

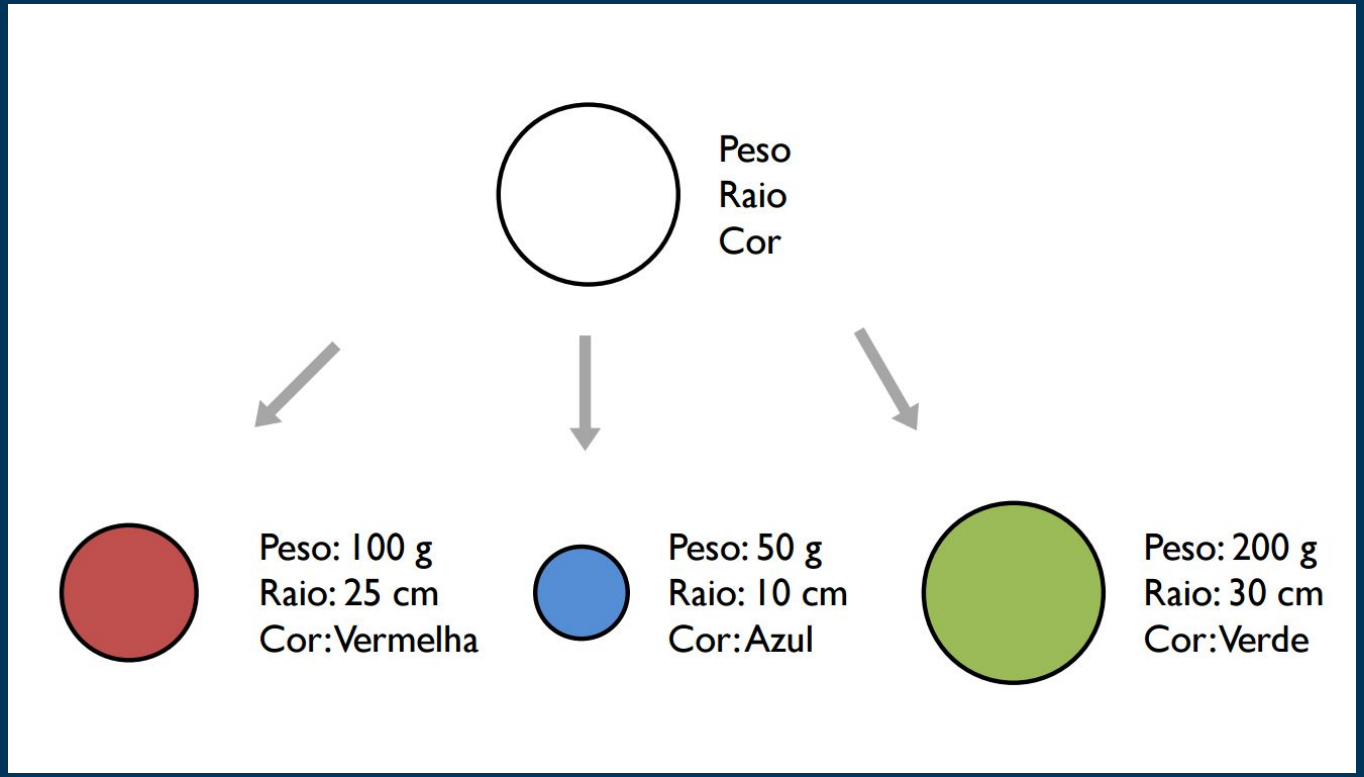
?>

Objetos

```php
// Elefante(Idade, Peso)
$dumbo = new Elefante(2, 400);

$chico = new Elefante(3, 389);

$dumbo->mostrarRiscoDeDoenca();
$chico->mostrarRiscoDeDoenca();
```

?>

Classe X Objetos



?>

Classes - Componentes

- Declarações de atributos
  - Atributos de classe
  - Atributos de instância

- Declarações de métodos
  - Métodos construtores (inicialização)
  - Métodos de classe Métodos de instância

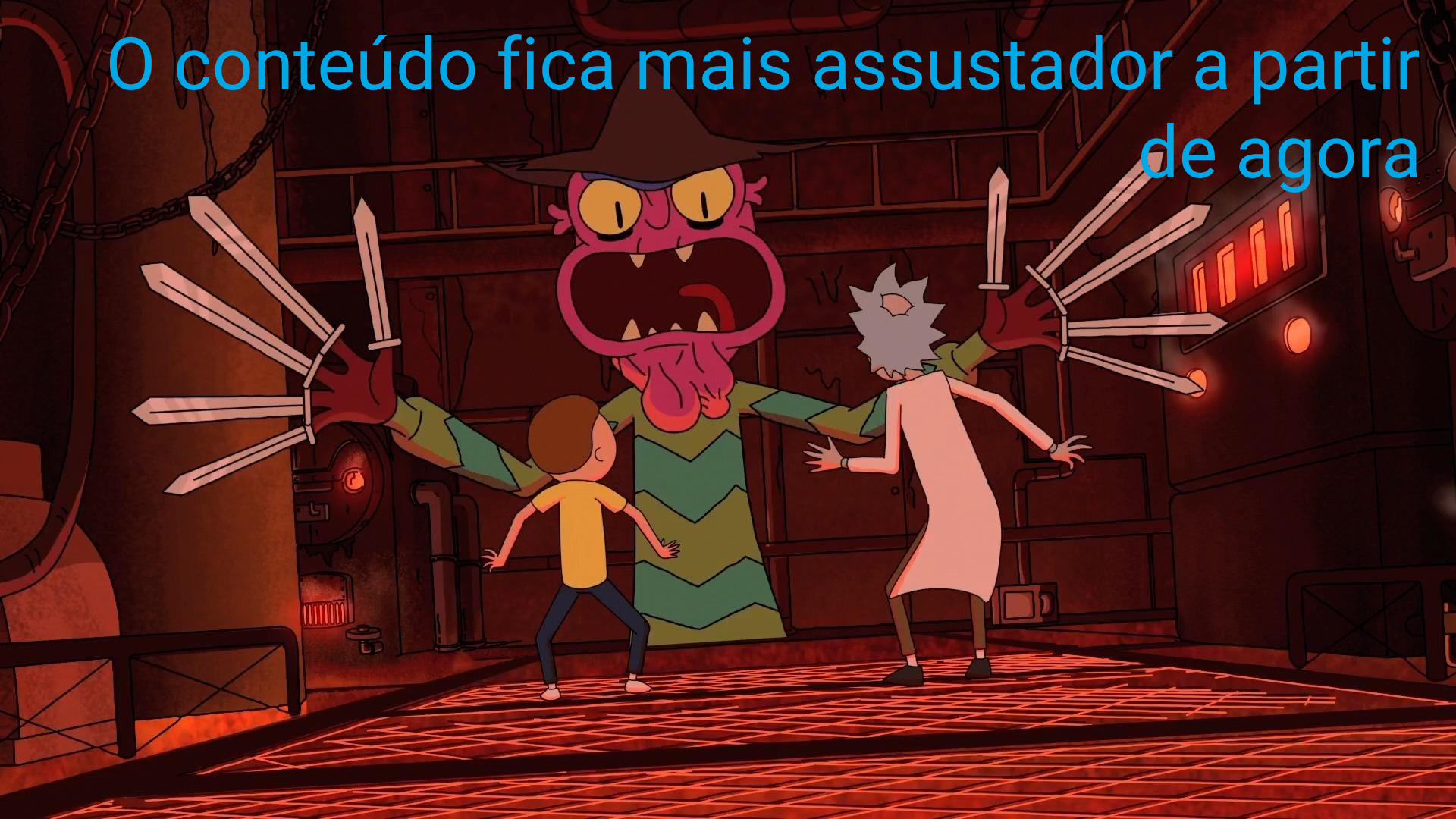
?>

Classes - Visibilidade

- Private
  - Visível apenas para a classe
- Protected
  - Visível para a classe e para as filhas
- Public
  - Visível para todos

?>


Se você entendeu
tudo até aqui,
parabéns


O conteúdo fica mais assustador a partir de agora


Mas elas vão te
ajudar um dia!

Métodos e Constantes Estáticas

```php
class Math
{
    public static PI = 3.1416;

    public static areaDoCirculo($radius) {
        return self::PI * self::PI * $radius;
    }
}

$myPI = Math::PI;
echo $myPI;

echo Math::areaDoCirculo(4);
```

Sobrecarga

- Diferenciáveis pela lista de parâmetros
  - A ordem dos tipos de parâmetros é importante

- Não são diferenciáveis pelo tipo de retorno
  - Podem possuir diferentes tipos de retorno desde que possuam diferentes parâmetros

?>

Sobrecarga

```php
class Ponto {
    private $x;
    private $y;

    public function setXY($x, $y) {
        $this->x = $x;
        $this->y = $y;
    }

    public function setXY($xy) {
        $this->x = $xy;
        $this->y = $xy;
    }
}
```

?>

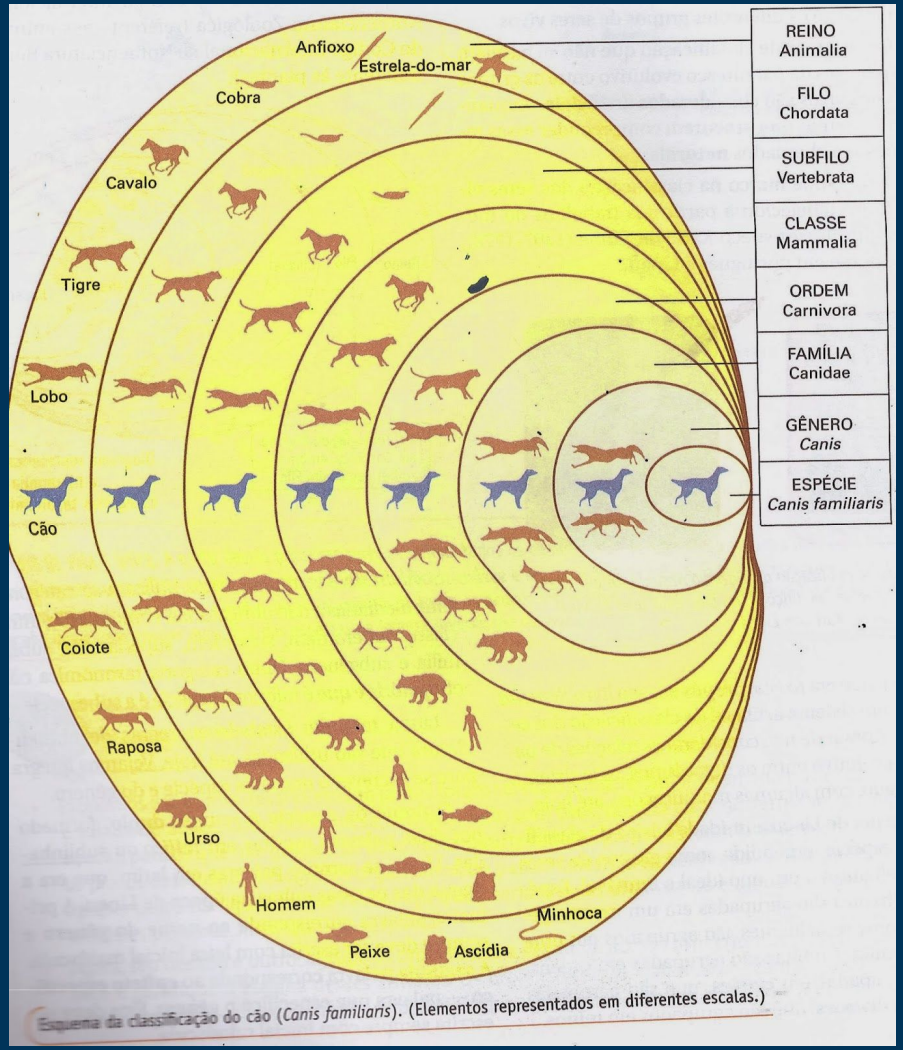
Herança

- Técnica para reutilizar características de uma classe na definição de outra classe

- Determinação de uma hierarquia de classes
  Terminologias relacionadas à Herança
  - Classes mais genéricas: superclasses (pai)
  - Classes especializadas: subclasses (filha)

?>

Herança



Esquema da classificação do cão (*Canis familiaris*). (Elementos representados em diferentes escalas.)

?>

Herança

- Superclasses
  - Devem guardar membros em comum
- Subclasses
  - Acrescentam novos membros (especializam)
- Componentes facilmente reutilizáveis
  - Facilita a extensibilidade do sistema
  - A subclasse deve utilizar o operador extends

Herança

```php
class Person {
    private $idade;

    public function setIdade($idade) {
        $this->idade = $idade;
    }
}

class Adulto extends Person {
    // Roda quuando o Objeto é criado
    function __construct()
    {
        $this->setIdade(21);
    }
}
```

?>

Sobrescrita

```php
class Pessoa {
    public function apresentacao()
    {
        echo "Oi, eu sou uma PESSOA";
    }
}

class Estudante extends Pessoa
{
    public function apresentacao()
    {
        echo "Oi, eu sou um estudante";
    }
}
```

?>

Classe Abstrata

- Uma Classe que não pode ser instanciada

- Define um conjunto de métodos
  - Totalmente implementados
  - Parcialmente implementados (contrato)
    - Declarados como abstract
    - Devem ser implementados nas classes derivadas

?>

Classe Abstrata

```php
abstract class Ponto {
    private $x;
    private $y;

    public function setXY($x, $y) {
        $this->x = $x;
        $this->y = $y;
    }
    public abstract function setXY($xy);
}

class Ponto2D extends Ponto {
    public function setXY($xy)
    {
        $this->setXY($xy, $xy);
    }
}
```

?>

Interfaces

- Possui unicamente o papel de um contrato
  - Garantia de que uma determinada Classe irá implementar um certo grupo de métodos
- Declarada da mesma forma que uma Classe
- Não suportam métodos estáticos
- Uma Classe **implementa** uma Interface

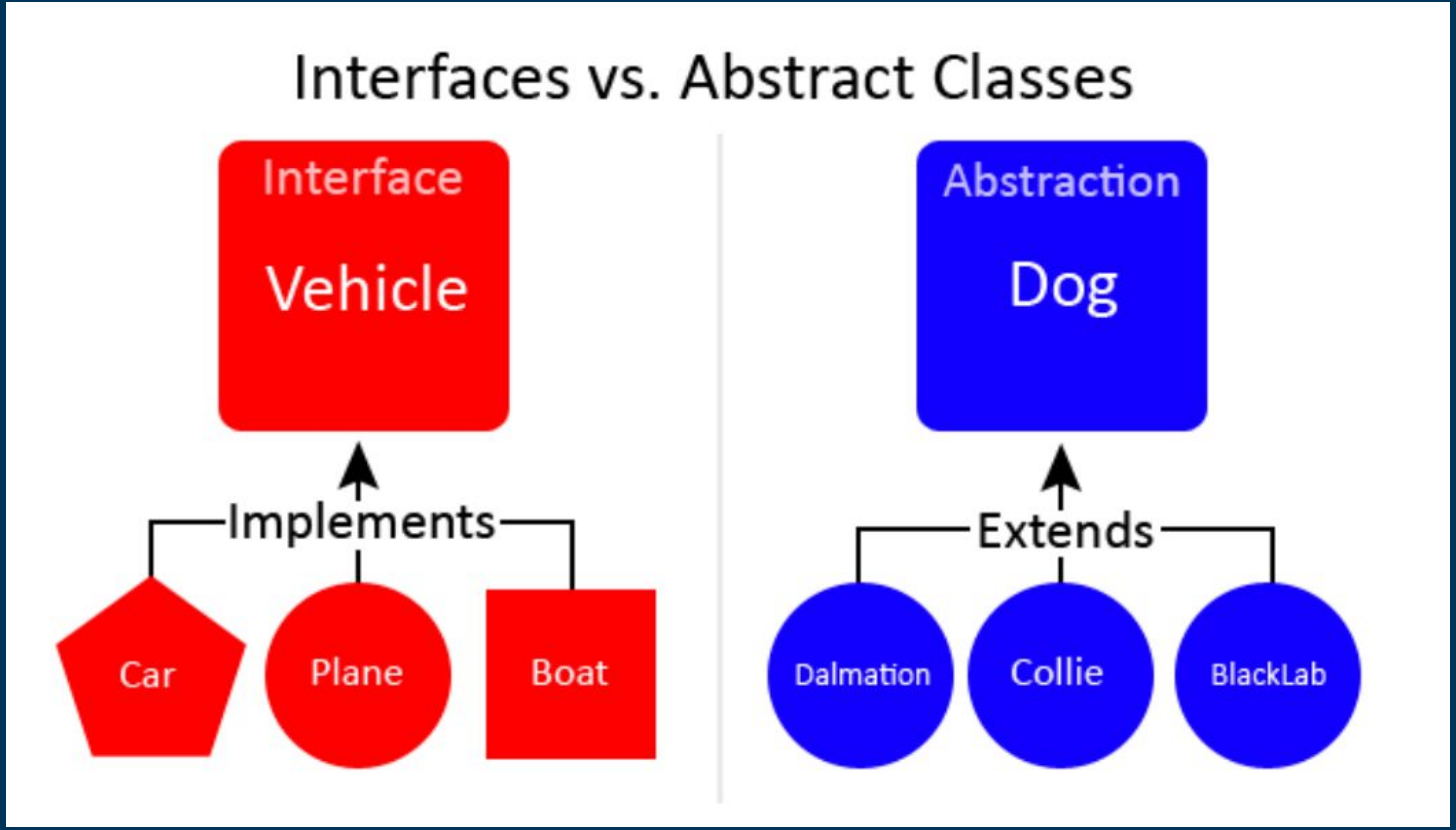
?>

Interfaces

```php
interface Imprimivel {
    public function imprimirValores();
}

class Ponto implements Imprimivel{
    private $x;
    private $y;

    public function setXY($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    public function FunctionName($value='')
    {
        printf("[%d, %d]", $this->x, $this->y);
    }
}
```

?>

Interfaces x Classes Abstratas



?>

Polimorfismo?

?>

Heurística

- Abstração
  - Ignorar detalhes muito específicos
  - Tratar os objetos como se fossem iguais
- Decomposição
  - Separar o problema em partes
- Projeção
  - Considerar diferentes visões sobre uma parte
- Modularização
  - Optar por estruturas estáveis no tempo

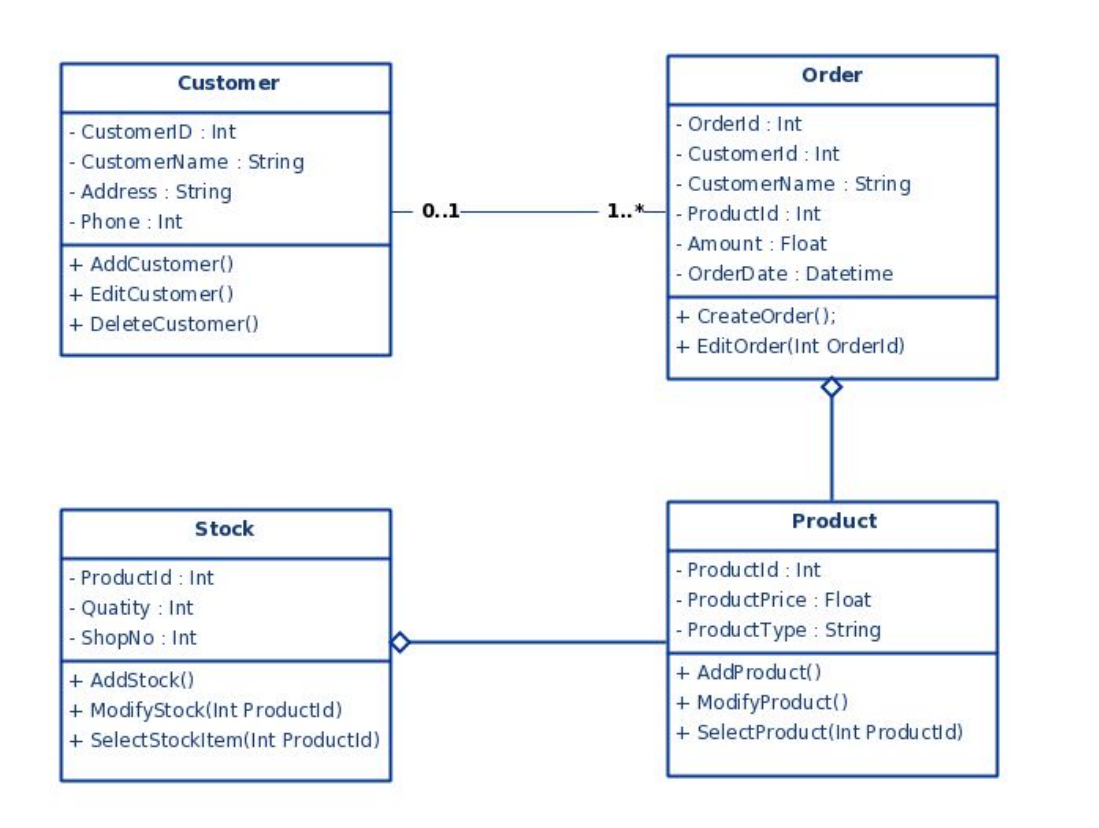
?>

UML

- Unified
  - Agrupamento de diferentes abordagens
- Modeling
  - Etapa fundamental (parte criativa)
  - Concepção e desenvolvimento de software
- Language
  - Disponibiliza uma notação formal

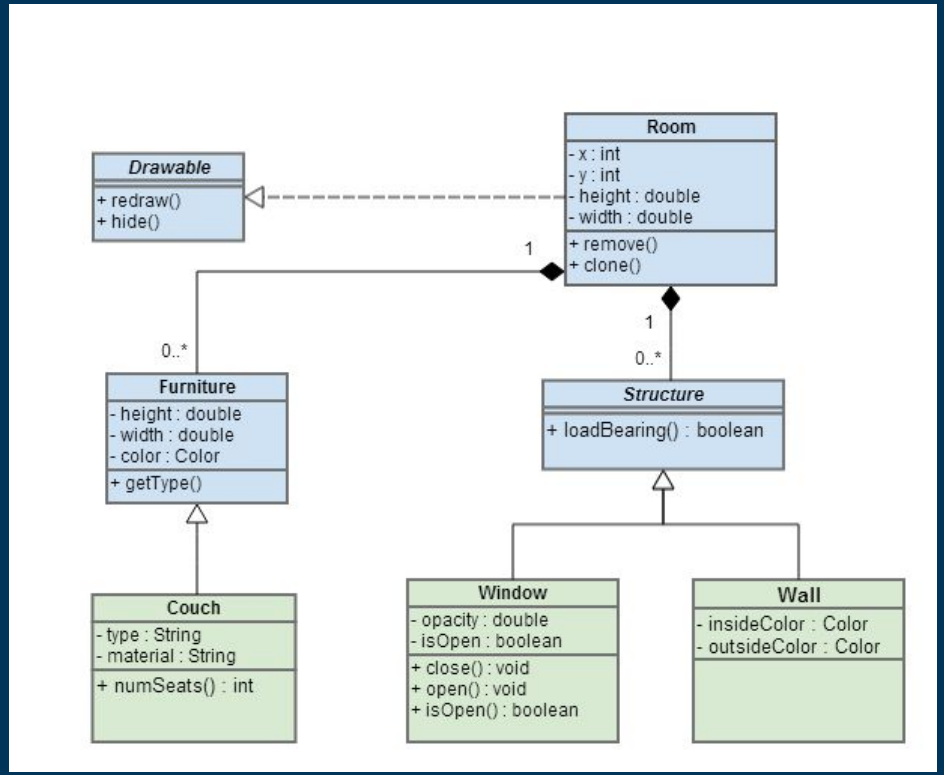
?>

UML - Diagrama de Classe



?>

UML - Diagrama de Classe



?>

Padrões de Projeto

- Padrões de projeto não são algoritmos!
- Padrões de projeto não são componentes!
  - Design Patterns != Frameworks
  - Framework Estrutura de suporte (códigos)
- Apresentam uma solução para problemas
- Funcionam como um “Livro de Receitas”

?>

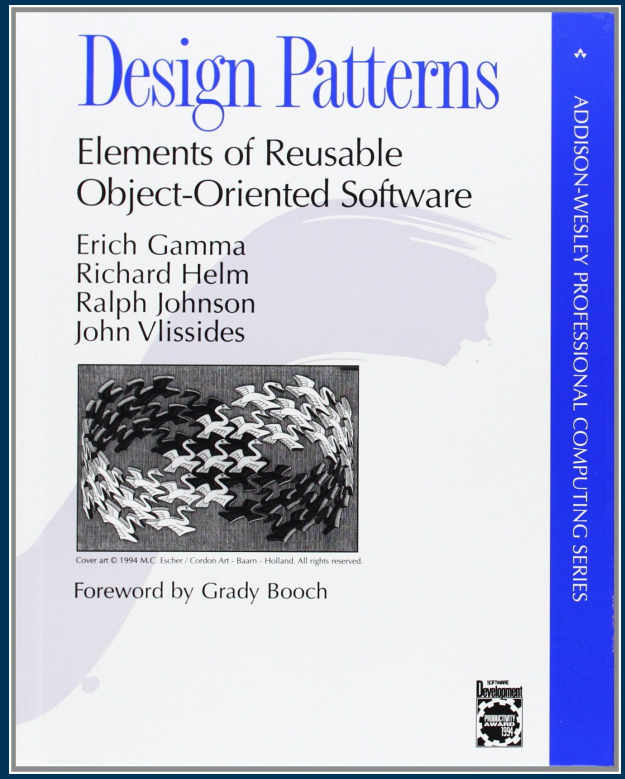
Padrões de Projeto

- Aprender com as experiências dos outros
  - Útil especialmente para iniciantes!
- Soluções bem testadas e documentadas
- Utilizam corretamente os conceitos de OO
  - Abstração, herança, polimorfismo, modularidade, composição, para construir código reutilizável, eficiente, de alta coesão e baixo acoplamento

?>

Padrões de Projeto

"Descrição de uma solução customizada para resolver um problema genérico de projeto em um contexto específico. [...] Um padrão de projeto dá nome, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la reutilizável."



?>

Padrões de Projeto

Criacionais	Estruturais	Comportamentais
• Abstract Factory • Builder • Factory Method * • Prototype • Singleton	• Adapter * • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of Responsibility • Command • Interpreter * • Iterator • Mediator • Memento • Observer • State • Strategy • Template Method * • Visitor

Padrões de Projeto - Factory

?>

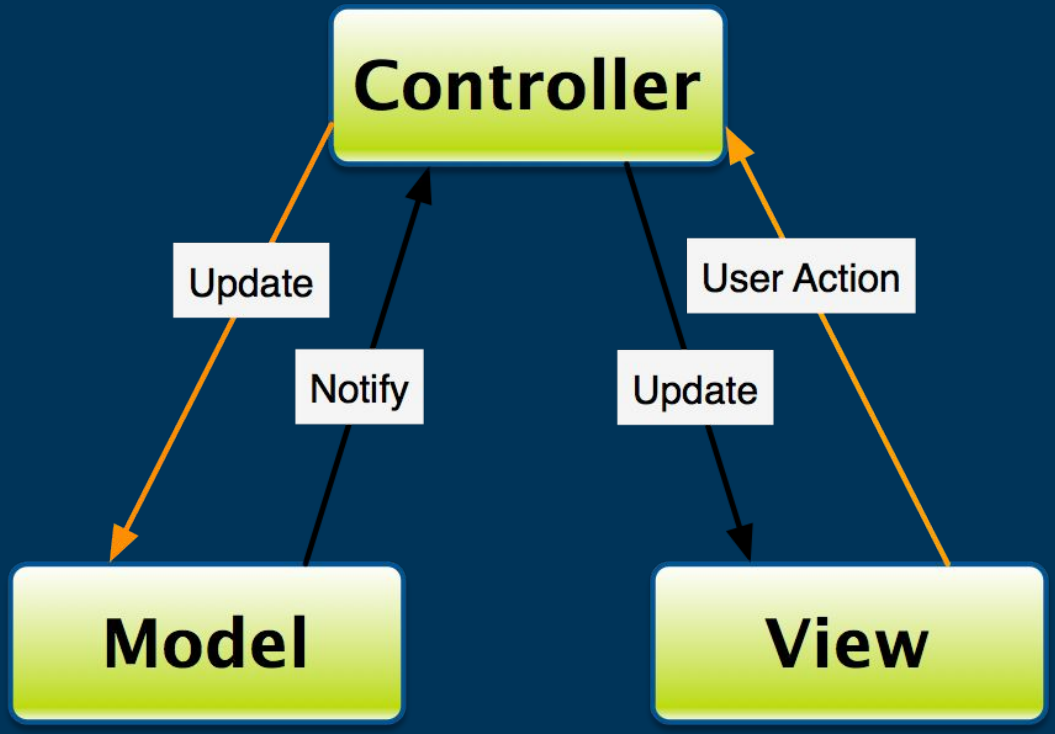
Padrões de Projeto

- Modelo-Visão-Controlador (MVC)
  - Modelo: armazena um conteúdo
  - Visão: exibe o conteúdo
  - Controlador: processa entradas do usuário

- Objetivo
  - Separar a apresentação da informação da interação do usuário com a informação

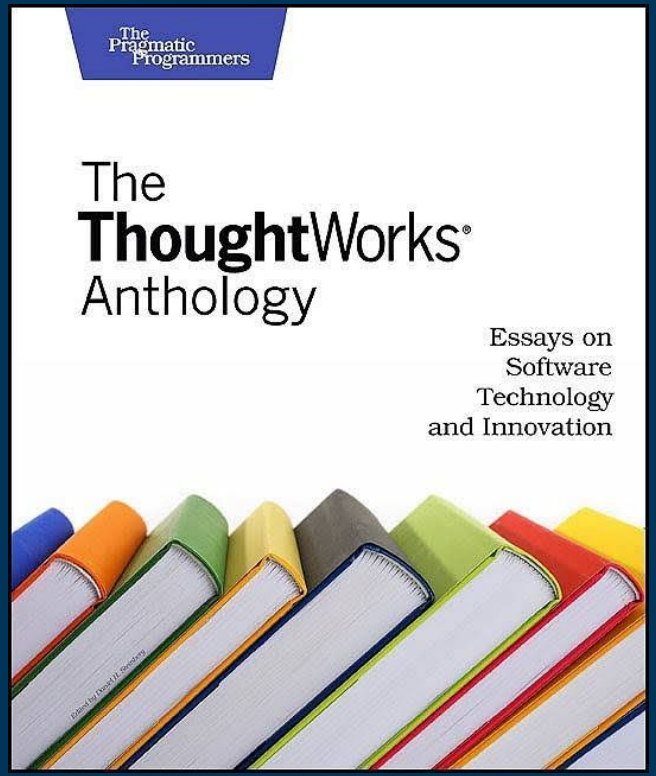
?>

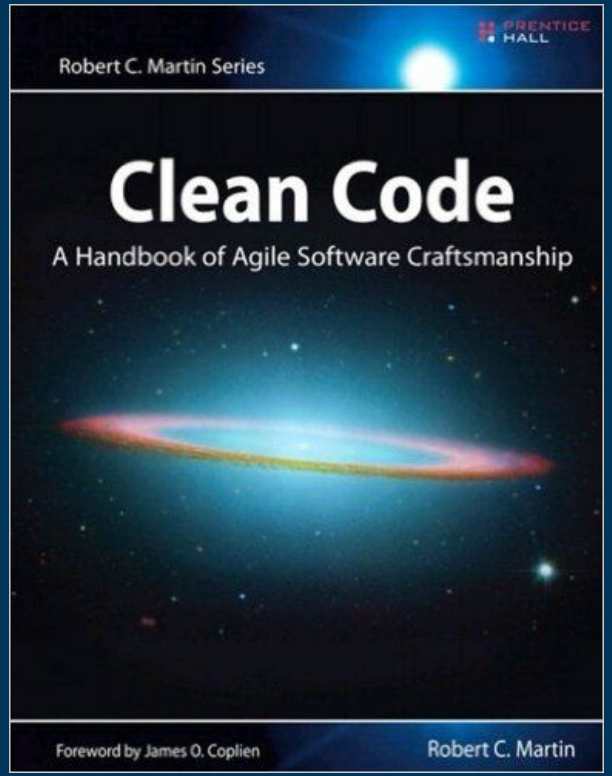
MVC

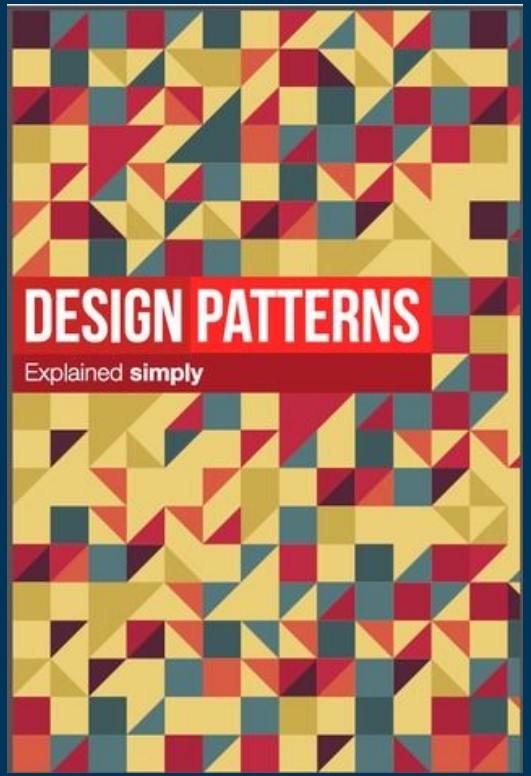


?>

Referências







Considerações Finais

Esses conceitos só são aprendidos com prática

Essa aula foi extremamente corrida

Aula de Programação Modular (DCC052)

?>

lucasmbastos@dcc.ufmg.br

lucasmbastos

?>

Referências

Slides do Professor Douglas Macharet da disciplina de Programação Modular

?>