Programação Orientada a Objetos com PHP
Exemplos de Herança e Sobrescrita

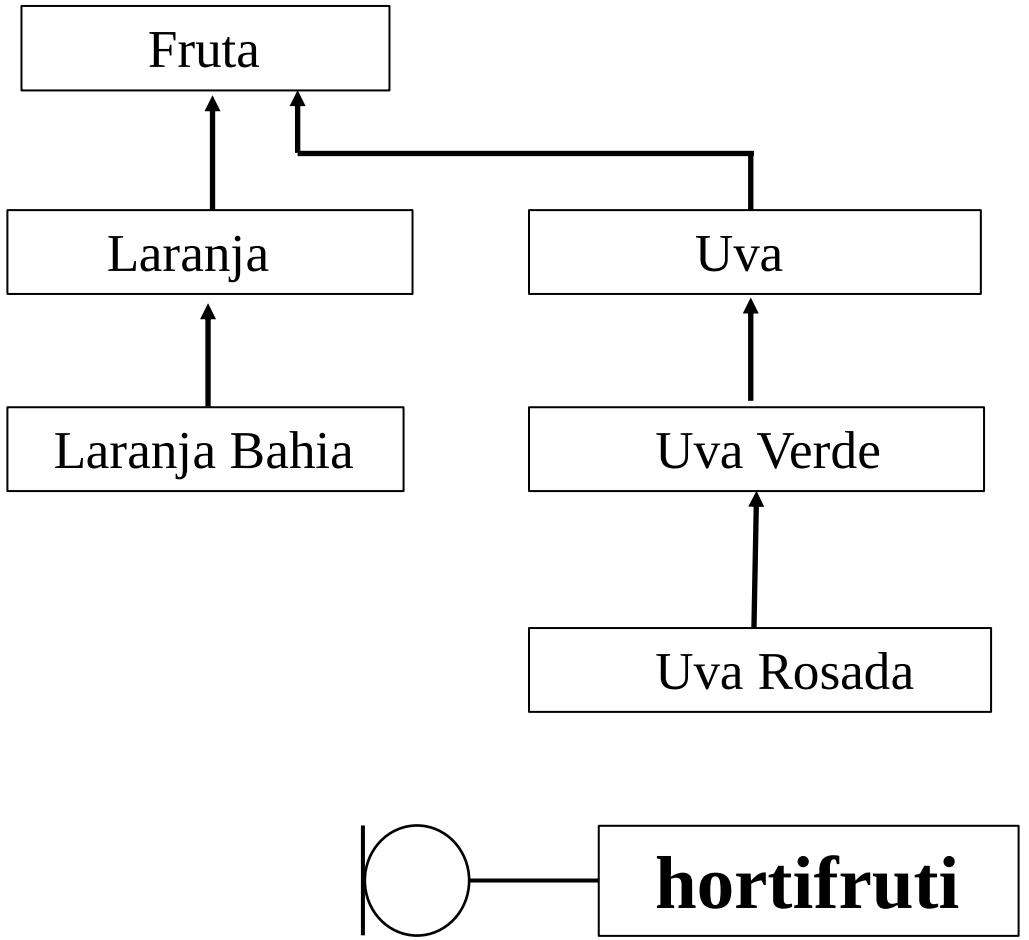
profº Mauricio Conceição Mario

Herança

- Herança é uma propriedade originada a partir de uma ligação hierárquica entre classes, onde classes que estão em camadas inferiores dessa ligação hierárquica podem herdar atributos e comportamentos das classes que estão em hierarquia superior. O uso de Herança permite, portanto, o reaproveitamento de propriedades de uma classe.
- A classe que se encontra no topo da hierarquia é denominada de ***superclasse*** ou ***classe-mãe***, enquanto que as classes que estão na hierarquia inferior são chamadas de ***subclasses*** ou ***classes-filhas***.

(Dall'Oglio, 2014)

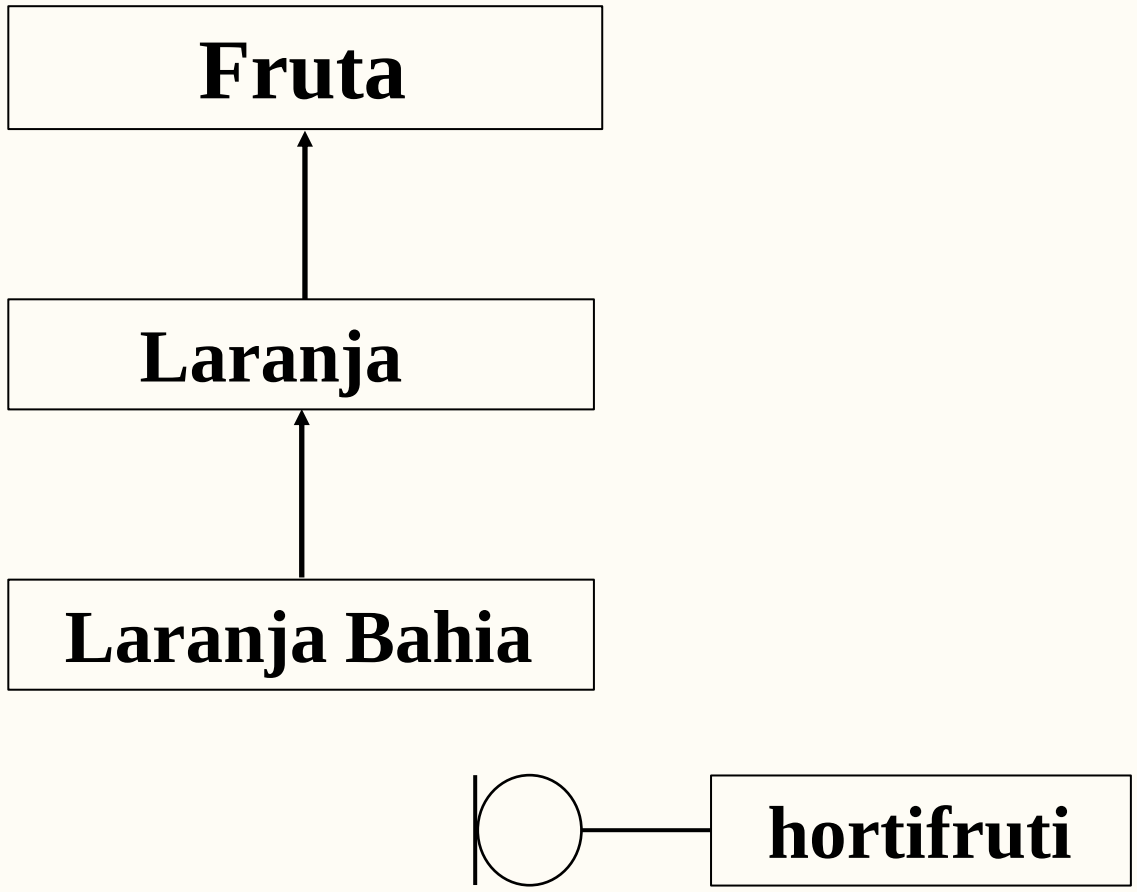
Exercício 10: De acordo com o diagrama de classes ao lado, inserir na relação de herança relativa à abstração das frutas, a classe “Uva” (no mesmo nível hierárquico da classe “Laranja”), e as classes-filhas “Uva Verde” e “Uva Rosada”.
Utilizar a aplicação “hortifruti” para inserir e retornar atributos pertinentes a estas novas classes da aplicação. Fazer a simulação no navegador.



Exemplo: superclasse ou classe-mãe → Fruta.class.php

Fruta.class.php | Laranja.php | Laranja_Bahia.php | hortifruti.php

```php
<?php
class Fruta
{
    protected $arvore;
    protected $nome_fruta;
    protected $preço;

    public function __construct($detalhe){
        print "detalhe da fruta: <br> {$detalhe}<br>\n";
    }

    public function set_arvore($árvore){
        $this->árvore = $árvore;
    }

    public function get_arvore(){
        return $this->árvore;
    }

    public function set_nome_fruta($nome_fruta){
        $this->nome_fruta = $nome_fruta;
    }

    public function get_nome_fruta(){
        return $this->nome_fruta;
    }

    public function set_preço($preço){
        $this->preço = $preço;
    }

    public function get_preço(){
        return $this->preço;
    }
}
?>
```



cliente_1.php | Uva.php | horário.php | Acessa_construtor.php | Fruta.clas

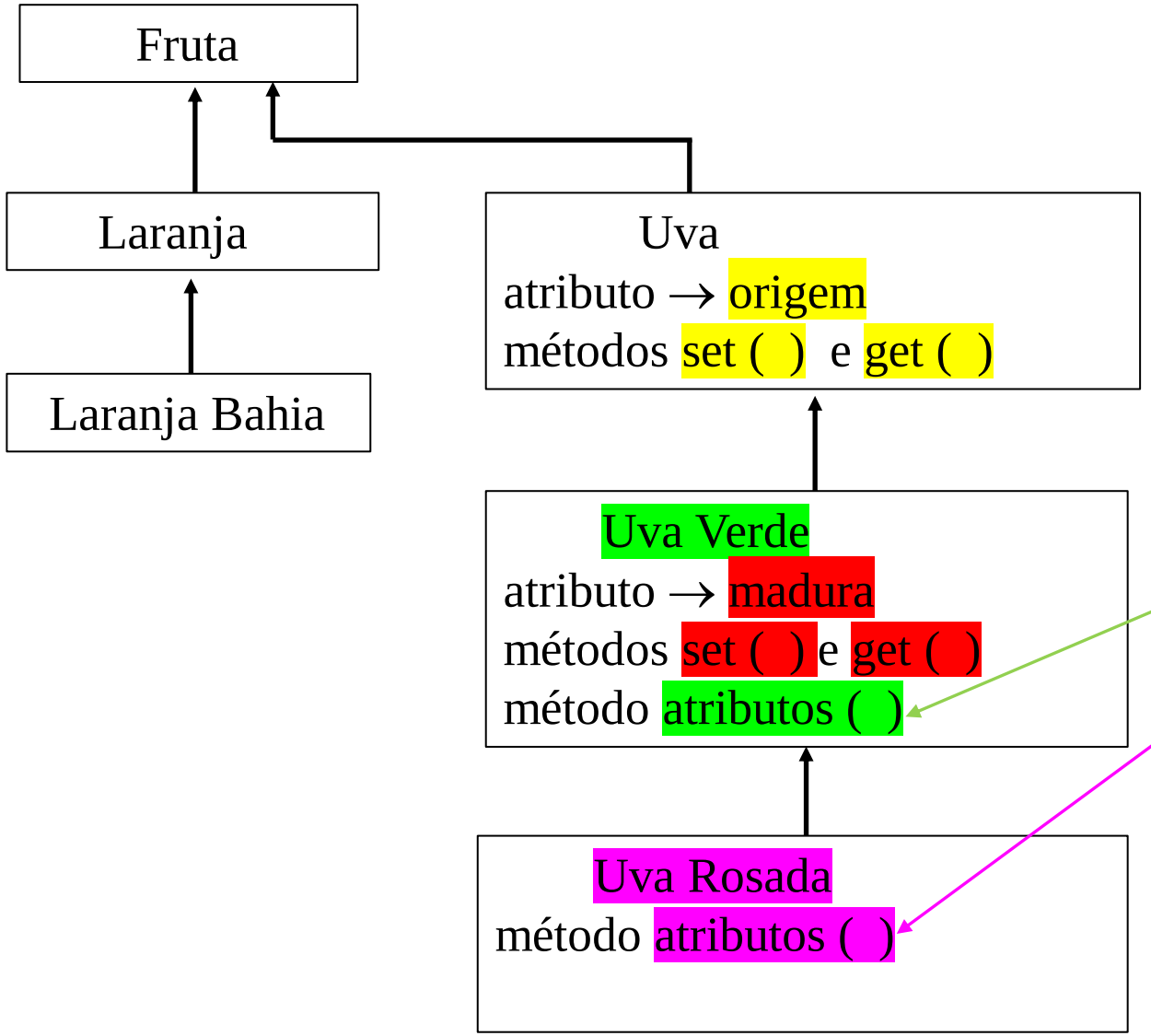
```php
<?php
class Uva extends Fruta
{
    protected $origem;

    public function __construct($f){
        parent::__construct($f);
    }

    public function set_origem($origem){
        $this->origem = $origem;
    }

    public function get_origem(){
        return $this->origem;
    }
}
?>
```

cliente_1.php | Uva.php | Uva_Rosada.php | Uva_Verde.php | horá

```php
<?php
class Uva_Rosada extends Uva_Verde

{
    public function __construct($e){
        parent::__construct($e);
        }
    public function atributos(){
        echo nl2br("na hierarquia desta herança
        a classe Uva_Rosada aparece como classe
        filha de Uva_Verde e pode acessar os
        atributos origem e madura\n");
        }
}
?>
```

cliente_1.php | Uva.php | Uva_Verde.php | horário.php | Acessa_construt

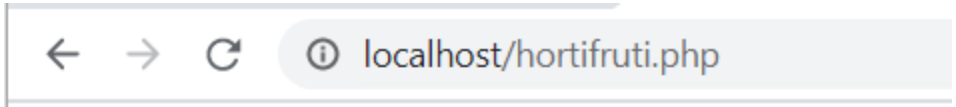
```php
<?php
class Uva_Verde extends Uva

{
    protected $madura;
    public function __construct($e){
        parent::__construct($e);
        }

    public function set_madura($madura){
        $this->madura = $madura;
    }

    public function get_madura(){
        $this->madura;
        if ($this->madura == True)
            return "sim";
        else return "não";
    }

    public function atributos(){
        echo nl2br("na hierarquia desta herança
        a classe Uva_Verde aparece no mesmo nível
        da classe Laranja_Bahia e pode acessar o
        atributo origem\n");
        }

}
?>
```



acesso ao método atributos ( ) é feito de acordo com a instância, ou à classe Uva Verde ou à classe Uva Rosada.
→ caracteriza o conceito de *Polimorfismo*: métodos com a mesma assinatura (identificação e forma) porém com funcionalidades diferentes.

cliente_1.php | Uva.php | Uva_Rosada.php | hortifruti.php | Uva_Verde.php

```php
<?php
require_once 'Fruta.class.php';
require_once 'Laranja.php';
require_once 'Laranja_Bahia.php';
require_once 'Uva.php';
require_once 'Uva_Verde.php';
require_once 'Uva_Rosada.php';

$f = new Laranja_Bahia("laranja bahia está bem doce");
$f->set_arvore("laranjeira");
$f->set_nome_fruta("laranja bahia");
$f->set_preço(8.22);
$f->set_cor("amarelo");
echo nl2br("dá na árvore: ".$f->get_arvore()."\n");
echo nl2br("fruta: ".$f->get_nome_fruta()."\n");
echo nl2br("preço = $".$f->get_preço()." a dúzia"."\n");
echo nl2br("cor: ".$f->get_cor()."\n");
echo nl2br("\n");

$g = new Laranja("laranja lima está azeda");
$g->set_arvore("laranjeira");
$g->set_nome_fruta("laranja lima");
$g->set_preço(6.05);
$g->set_cor("amarelo-esverdeada");
echo nl2br("dá na árvore: ".$g->get_arvore()."\n");
echo nl2br("fruta: ".$g->get_nome_fruta()."\n");
echo nl2br("preço = $".$g->get_preço()." a dúzia"."\n");
echo nl2br("cor: ".$g->get_cor()."\n");
echo nl2br("\n");

$h = new Fruta("maçã argentina está bem grande ");
$h->set_arvore("macieira");
$h->set_nome_fruta("maçã");
$h->set_preço(12.66);
echo nl2br("dá na árvore: ".$h->get_arvore()."\n");
echo nl2br("fruta: ".$h->get_nome_fruta()."\n");
echo nl2br("preço = $".$h->get_preço()." a dúzia"."\n");
echo nl2br("\n");


$uv = new Uva_Verde("uva verde boa para salada de frutas");
$uv->atributos();
$uv->set_madura(True);
$uv->set_arvore("videira");
$uv->set_nome_fruta("uva verde");
$uv->set_preço(14.87);
$uv->set_origem("Vinhedo");
echo nl2br("dá na árvore: ".$uv->get_arvore()."\n");
echo nl2br("fruta: ".$uv->get_nome_fruta()."\n");
echo nl2br("está madura?  ".$uv->get_madura()." "."\n");
echo nl2br("preço = $".$uv->get_preço()." a dúzia"."\n");
echo nl2br("cidade da uva: ".$uv->get_origem()."\n");
echo nl2br("\n");

$ur = new Uva_Rosada("uva rosada muito saborosa");
$ur->atributos();
$ur->set_madura(False);
$ur->set_arvore("videira");
$ur->set_nome_fruta("uva rosada");
$ur->set_preço(11.34);
$ur->set_origem("Jundiaí");
echo nl2br("dá na árvore: ".$ur->get_arvore()."\n");
echo nl2br("fruta: ".$ur->get_nome_fruta()."\n");
echo nl2br("está madura?  ".$ur->get_madura()." "."\n");
echo nl2br("preço = $".$ur->get_preço()." a dúzia"."\n");
echo nl2br("cidade da uva: ".$ur->get_origem()."\n");
echo nl2br("\n");

?>
```



detalhe da fruta:
laranja bahia está bem doce
dá na árvore: laranjeira
fruta: laranja bahia
preço = $8.22 a dúzia
cor: amarelo

detalhe da fruta:
laranja lima está azeda
dá na árvore: laranjeira
fruta: laranja lima
preço = $6.05 a dúzia
cor: amarelo-esverdeada

detalhe da fruta:
maçã argentina está bem grande
dá na árvore: macieira
fruta: maçã
preço = $12.66 a dúzia

detalhe da fruta:
uva verde boa para salada de frutas
na hierarquia desta herança
a classe Uva_Verde aparece no mesmo nível
da classe Laranja_Bahia e pode acessar o
atributo origem
dá na árvore: videira
fruta: uva verde
está madura? sim
preço = $14.87 a dúzia
cidade da uva: Vinhedo

detalhe da fruta:
uva rosada muito saborosa
na hierarquia desta herança
a classe Uva_Rosada aparece como classe
filha de Uva_Verde e pode acessar os
atributos origem e madura
dá na árvore: videira
fruta: uva rosada
está madura? não
preço = $11.34 a dúzia
cidade da uva: São Roque

Exercícios:

11. Inserir na aplicação “hortifruti.php” uma variável do tipo “Uva”, e acessar todos os atributos e métodos que podem ser acessados por esta classe, mostrando os valores atribuídos no navegador.

12. Criar uma aplicação, utilizando as hierarquias de herança, para representar algumas espécies de animais.

Exemplo: superclasse ou classe-mãe → Conta_Bancária.class.php.

Conta_Bancária.class.php

```php
<?php
 class Conta_Bancária
{
    protected $agencia;
    protected $conta;
    protected $saldo;

    public function __construct($agencia, $conta, $saldo)
    {
        $this->agencia = $agencia;
        $this->conta   = $conta;
        if ($saldo >= 0) {
            $this->saldo   = $saldo;
        }
    }
    public function getInfo()
    {
        return "Agência: {$this->agencia}, Conta: {$this->conta}";
    }
    public function depositar($quantia)
    {
        if ($quantia > 0) {
            $this->saldo += $quantia;
        }
    }

    public function getSaldo()
    {
        return $this->saldo;
    }
}
?>
```

(Dall'Oglio, 2014)

Subclasse ou classe-filha: Conta_Corrente.php.

Conta_Bancária.class.php | Consignado.php | Conta_Poupanca.php | aplica_herança2.php | Conta_Corrente.php

```php
<?php
class Conta_Corrente extends Conta_Bancária
{
    protected $limite;

    public function __construct($agencia, $conta, $saldo, $limite)
    {
        parent::__construct($agencia, $conta, $saldo);
        $this->limite = $limite;
    }

    public function retirar($quantia)
    {
        if ( ($this->saldo + $this->limite) >= $quantia ) {
            $this->saldo -= $quantia; // retirada permitida
        }
        else {
            return false; // retirada não permitida
        }
        return true;
    }
}
?>
```

parent::__construct (...) → acesso ao método construtor da superclasse

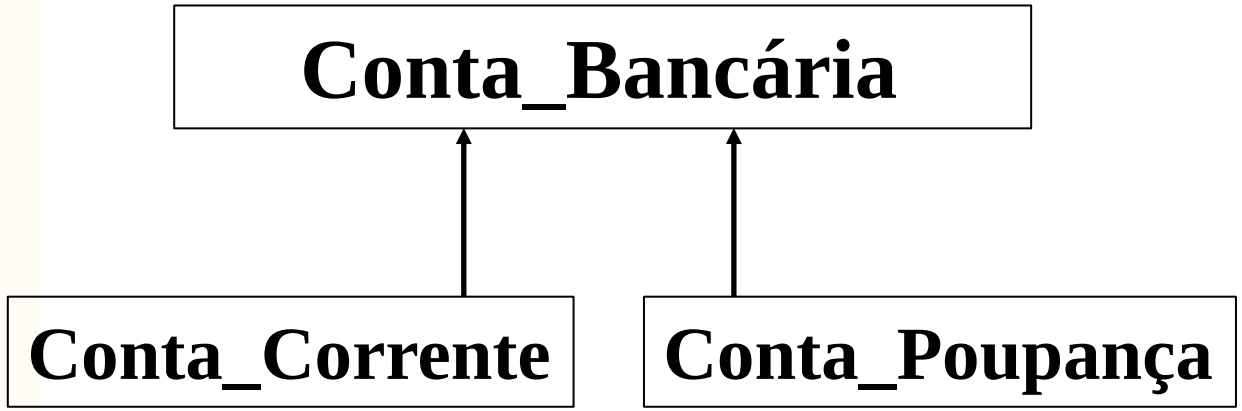
(Dall'Oglio, 2014)

Subclasse ou classe-filha: Conta_Poupança.php.

Conta_Bancária.class.php | Consignado.php | Conta_Poupanca.php | aplica_herança2.php

```php
<?php
 class Conta_Poupanca extends Conta_Bancária
{

    public function __construct($agencia, $conta, $saldo)
    {
        parent::__construct($agencia, $conta, $saldo);

    }

    function retirar($quantia)
    {
        if ($this->saldo >= $quantia) {
            $this->saldo -= $quantia;
        }
        else {
            return false; // retirada não permitida
        }
        return true; // retirada permitida
    }
}
?>
```

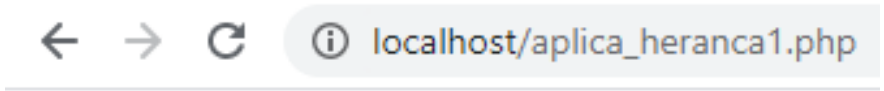
Diagrama de Classes → hierarquia:



(Dall'Oglio, 2014)

aplicação: aplica_heranca1.php.

```php
<?php
require_once 'Conta_Bancária.class.php';
require_once 'Conta_Poupanca.php';
require_once 'Conta_Corrente.php';

// criação dos objetos
$contas = array();
$contas[0] = new Conta_Corrente(6677, "CC.1234.56", 300, 400);
$contas[1] = new Conta_Poupanca(6678, "PP.1234.57", 100);

// percorre as contas
foreach ($contas as $key => $conta)
{
    print "Conta: {$conta->getInfo()} <br>\n";
    print "    Saldo atual: {$conta->getSaldo()} <br>\n";
    $conta->depositar(200);
    print "    Depósito de: 200 <br>\n";
    print "    Saldo atual: {$conta->getSaldo()} <br>\n";

    if ($conta->retirar(700)) {
        print "    Retirada de: 700 <br>\n";
    }
    else
    {
        print "    Retirada de: 700 (não permitida)<br>\n";
    }
    print "    Saldo atual: {$conta->getSaldo()} <br>\n";
}

?>
```



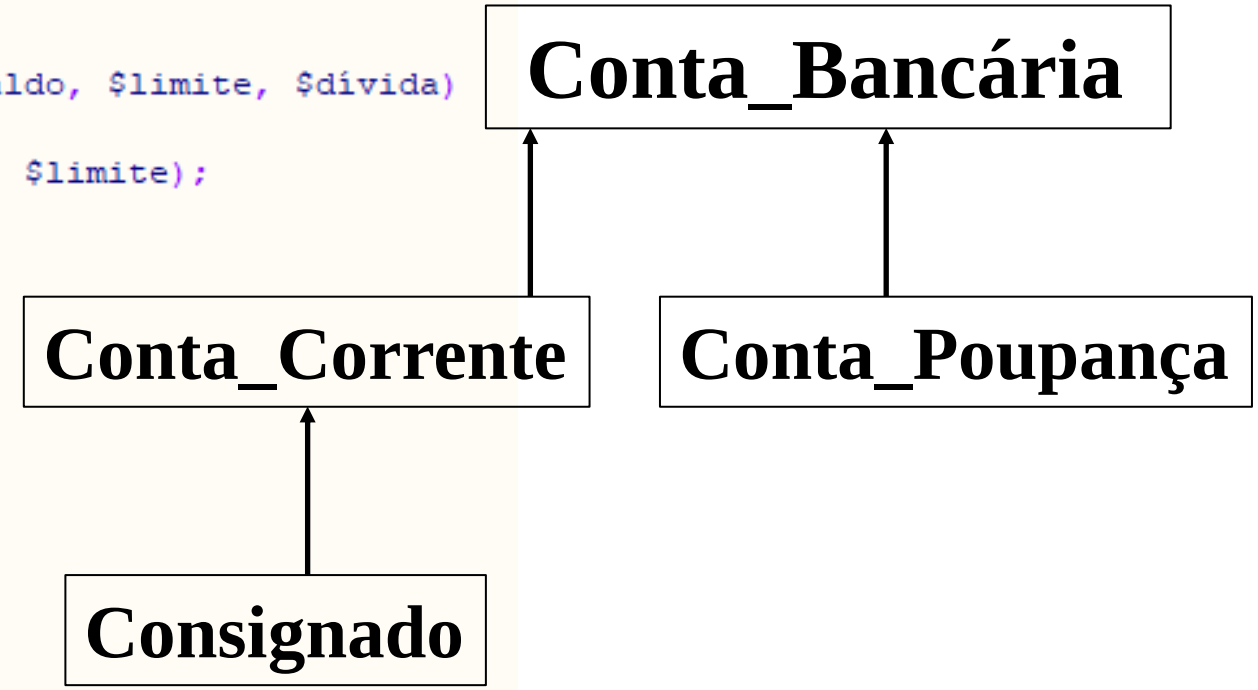
Conta: Agência: 6677, Conta: CC.1234.56
Saldo atual: 300
Depósito de: 200
Saldo atual: 500
Retirada de: 700
Saldo atual: -200
Conta: Agência: 6678, Conta: PP.1234.57
Saldo atual: 100
Depósito de: 200
Saldo atual: 300
Retirada de: 700 (não permitida)
Saldo atual: 300

Subclasse ou classe-filha: Consignado.php.

Conta.class.php | Consignado.php | Conta_Poupanca.php | aplica_herança2.php | Conta_Corrente.php | apli

```php
<?php
class Consignado extends Conta_Corrente
{
    protected $empréstimo;
    protected $dívida;

    public function __construct($agencia, $conta, $saldo, $limite, $dívida)
    {
        parent::__construct($agencia, $conta, $saldo, $limite);
        $this->dívida = $dívida;
    }

    public function set_empréstimo($empréstimo)
    {
        $this->empréstimo = $empréstimo;
    }

    public function get_empréstimo( )
    {
       return $this->empréstimo;
    }

    public function get_dívida( )
    {
       return $this->dívida;
    }
}
?>
```

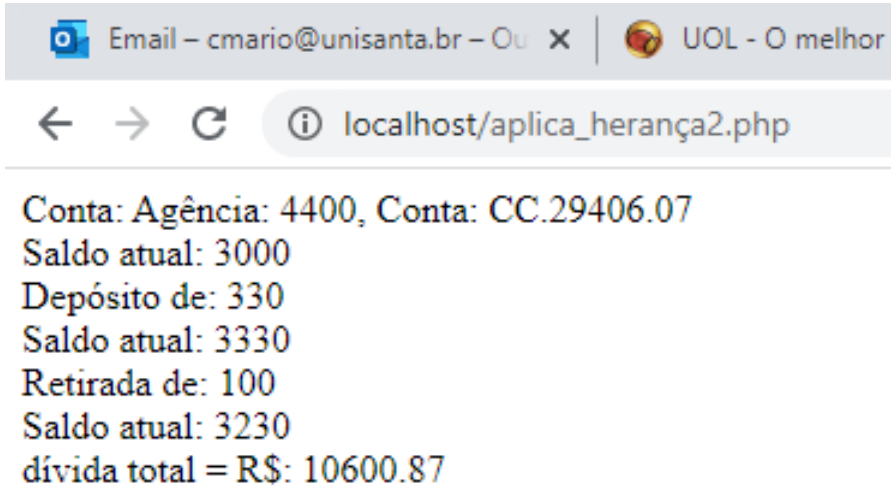
Diagrama de Classes → hierarquia:



aplicação: aplica_herança2.php.

Conta_Bancária.class.php | Consignado.php | Conta_Poupanca.php | aplica_herança2.php | Conta_Corren

```php
<?php
require_once 'Conta_Bancária.class.php';
require_once 'Conta_Poupanca.php';
require_once 'Conta_Corrente.php';
require_once 'Consignado.php';
// criação de objeto do tipo Consignado
$consig = new Consignado (4400, "CC.29406.07", 3000, 8800, 10000);

    print "Conta: {$consig->getInfo()} <br>\n";
    print "    Saldo atual: {$consig->getSaldo()} <br>\n";
    $consig->depositar(330);
    print "    Depósito de: 330 <br>\n";
    print "    Saldo atual: {$consig->getSaldo()} <br>\n";

    $consig->retirar(100);
        print "    Retirada de: 100 <br>\n";
        print "    Saldo atual: {$consig->getSaldo()} <br>\n";

    $consig->set_empréstimo(600.87);
    $dívida_total = ($consig->get_empréstimo()+ $consig->get_dívida());
    print " dívida total = R$: {$dívida_total} <br>\n";

?>
```



- Sobrescrita de métodos ou *Overriding*: o comportamento de um método da superclasse é modificado nas classes-filha, dando-lhe funcionalidades diferentes. Exemplo: acrescido o método contabilizar(..) nas classes da aplicação.

Conta_Bancária.class.php

```php
    public function contabilizar($saldo)
    { $this-> saldo += 60.55;
    print "Acesso pela classe Conta_Bancária <br>\n";
    }
}
?>
```

Conta_Corrente.php

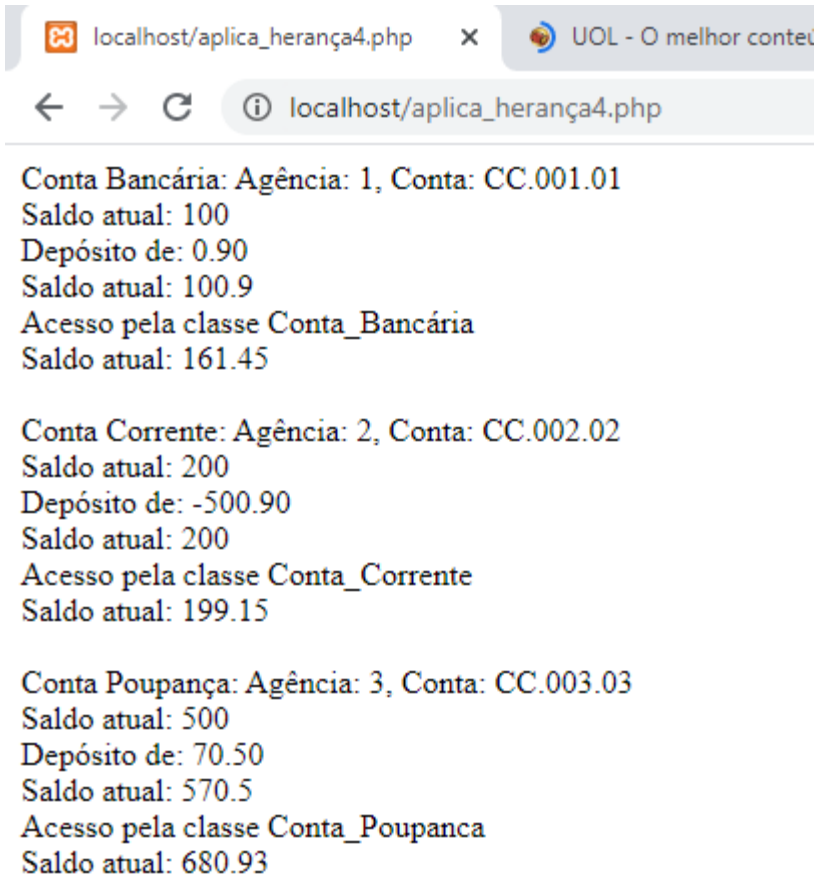
```php
    public function contabilizar($saldo)
    { $this-> saldo -= 0.85;
    print "Acesso pela classe Conta_Corrente <br>\n";
    }
}
?>
```

Conta_Poupanca.php

```php
    public function contabilizar($saldo)
    { $this-> saldo += 110.43;
    print "Acesso pela classe Conta_Poupanca <br>\n";
    }
}
?>
```

aplicação: aplica_herança4.php.

```php
<?php
require_once 'Conta_Bancária.class.php';
require_once 'Conta_Poupanca.php';
require_once 'Conta_Corrente.php';

/* criação de objetos tipo Conta_Corrente e Conta_Poupança
+ Conta_Bancária*/
$cb = new Conta_Bancária(000001, "CC.001.01", 100);
$cc = new Conta_Corrente(000002, "CC.002.02", 200, 200);
$cp = new Conta_Poupanca(000003, "CC.003.03", 500);

    print "Conta Bancária: {$cb->getInfo()} <br>\n";
    print "    Saldo atual: {$cb->getSaldo()} <br>\n";
    $cb->depositar(0.90);
    print "    Depósito de: 0.90 <br>\n";
    print "    Saldo atual: {$cb->getSaldo()} <br>\n";
    $cb->contabilizar(0.66);
    print "    Saldo atual: {$cb->getSaldo()} <br>\n";

    print "<br>\n Conta Corrente: {$cc->getInfo()} <br>\n";
    print "    Saldo atual: {$cc->getSaldo()} <br>\n";
    $cc->depositar(-500.90);
    print "    Depósito de: -500.90 <br>\n";
    print "    Saldo atual: {$cc->getSaldo()} <br>\n";
    $cc->contabilizar(-1.66);
    print "    Saldo atual: {$cc->getSaldo()} <br>\n";

    print "<br>\n Conta Poupança: {$cp->getInfo()} <br>\n";
    print "    Saldo atual: {$cp->getSaldo()} <br>\n";
    $cp->depositar(70.50);
    print "    Depósito de: 70.50 <br>\n";
    print "    Saldo atual: {$cp->getSaldo()} <br>\n";
    $cp->contabilizar(3.36);
    print "    Saldo atual: {$cp->getSaldo()} <br>\n";
?>
```



Exercício proposto:

13. Criar uma classe “Rendimentos.php”, filha de “Conta_Poupança.php”. Criar também a aplicação onde se possa acessar propriedades de Rendimentos e das classes que estão acessíveis a ela através da hierarquia de Herança. Considerar o exemplo “Consignado.php”. Criar na classe pedida uma versão do método sobrescrito.

Referências Bibliograficas

- *php* Programando com Orientação a Objetos
  Pablo Dall'Oglio - editora Novatec - 2014

- Desenvolvimento web com PHP e MySQL
  Evaldo Junior Bento – editora Casa do Código - 2018