uninorte

PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Edkallenn

Sistemas de Informação - Uninorte

Programação Orientada a Objetos

- **Prof. Edkallenn Lima**
- edkallenn@yahoo.com.br (somente para dúvidas)
- **Blogs**:
  - http://professored.wordpress.com (Computador de Papel – O conteúdo da forma)
  - http://professored.tumblr.com/ (Pensamentos Incompletos)
  - http://umcientistaporquinzena.tumblr.com/ (Um cientista por quinzena)
  - http://eulinoslivros.tumblr.com/ (Eu li nos livros)
  - http://linabiblia.tumblr.com/ (Eu li na Bíblia)
- **YouTube**:
  - https://www.youtube.com/user/edkallenn
  - https://www.youtube.com/channel/UC-pD2gnahhxUDVuTAA0DHoQ
- **Redes Sociais**:
  - http://www.facebook.com/edkallenn
  - http://twitter.com/edkallenn ou @edkallenn
  - https://plus.google.com/u/0/113248995006035389558/posts
  - Instagram: http://instagram.com/edkallenn ou @edkallenn
  - Foursquare: https://pt.foursquare.com/edkallenn
  - Pinterest: https://br.pinterest.com/edkallenn/
- **Telefones**:
  - 68 98401-2103 (CLARO e Whatsapp) e 68 3212-1211.

Os exercícios devem ser enviados SEMPRE para o e-mail:
edkevan@gmail.com
ou para o e-mail:
edkallenn.lima@uninorteac.com.br

Agenda

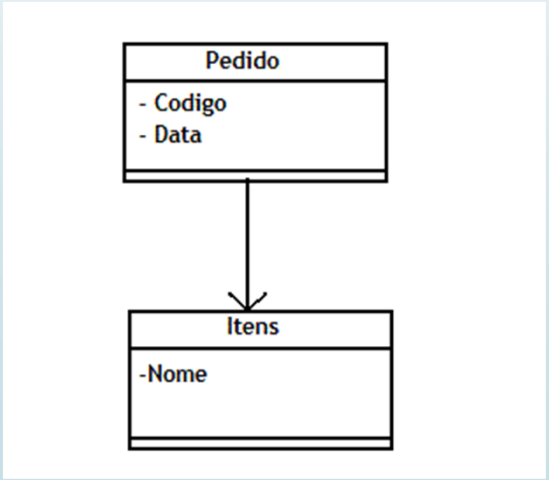
- Composição
- Herança
- Upcast e Downcast
- Polimorfismo
- Sobrescrita
- Associação
- Agregação, composição e dependência
- Interfaces

Conceitos relacionais

- Os conceitos relacionais são responsáveis por possibilitar a criação de classes a partir, ou com a ajuda, de outras classes.
- A seguir, veremos o que é herança, associação e interface.
- Também aprenderemos os subconceitos inerentes a cada um destes.

Composição

- **Composição**
- Estende uma classe e delega o trabalho para o objeto desta classe;
- Uma instância da classe existente é usada como componente da outra classe;
- **Tem um**;



- **Pedido** - Classe que contém uma instância da classe **Itens;**-
- Um pedido **TEM UM** Item;

Composição

- **Usando composição**
- 1- Os objetos que foram instanciados e estão contidos na classe que os instanciou são acessados somente através de sua interface;
  2- A composição pode ser definida dinamicamente em tempo de execução pela obtenção de referência de objetos a objetos de do mesmo tipo;
  3- A composição apresenta uma menor dependência de implementações;
  4- Na composição temos cada classe focada em apenas uma tarefa (princípio SRP);
  5- Na composição temos um bom encapsulamento visto que os detalhes internos dos objetos instanciados não são visíveis;

Herança

- O paradigma da Programação Orientada a Objetos (POO) trabalha determinando um retrato fidedigno dos objetos do mundo real.
- Logo, nada mais sensato e interessante que a existência de um conceito que denote a hierarquia existente entre os objetos reais.
- Tal conceito é de extrema importância e diferencia a linguagem de programação que, aliada à orientação a objetos, possibilita um melhor aproveitamento e reutilização do código por meio da ideia de herança

Herança

- O conceito de herança nada mais é do que uma possibilidade de representar algo que já existe no mundo real.
- Um exemplo clássico disto é quando na escola, na aula de ciências, estudamos sobre "classificação biológica".
- Nela a seguinte divisão é feita entre os seres vivos: Reino, Filo, Classe, Ordem, Família, Gênero, Espécie.
- Cada divisão mais baixa herda o que for necessário da divisão superior, e isto ocorre porque a mais baixa é um subtipo da divisão acima.
- Espécie herda de Gênero, que por sua vez herda de Família e assim por diante.

Herança

- No caso de nós seres humanos, nossa classificação dentro desta estrutura seria:
  - Reino: Animalia
  - Filo: Chordata
  - Classe: Mammalia
  - Ordem: Primates
  - Família: Hominidae
  - Gênero: Homo
  - Espécie: Homo Sapiens.
- No caso, Homo Sapiens herda de Homo, que por sua vez herda de Hominidae e assim por diante.
- Um exemplo mais simples é que todos nós herdamos algo de nossos pais, eles herdaram de nossos avós e assim por diante.

Herança

- Mas voltando a Orientação a Objeto, como podemos aplicar a herança?
- Já vimos conceitos de *Homo Sapiens*, *Homo* etc., e também que sempre que desejarmos representar um conceito do mundo real em uma linguagem orientada a objeto, o conceito de classe deve ser utilizado.
- Assim, na OO, quando desejarmos usar o conceito de herança, é necessário fazer uma classe herdar de outra classe.

Herança

- ***Herança é o relacionamento entre classes em que uma classe chamada de subclasse (classe filha, classe derivada) é uma extensão, um subtipo, de outra classe chamada de superclasse (classe pai, classe mãe, classe base).***
- ***Devido a isto, a subclasse consegue reaproveitar os atributos e métodos dela.***
- ***Além dos que venham a ser herdados, a subclasse pode definir seus próprios membros.***

Herança

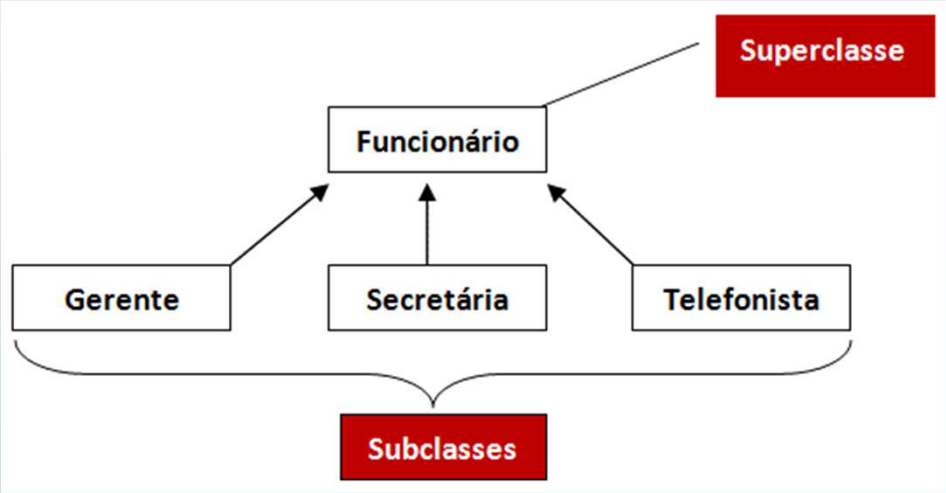
- A herança pode ocorrer em quantos níveis forem necessários.
- Porém, uma boa quantidade de níveis é de, no máximo, 4.
- Quanto mais níveis existirem, mais difícil de entender o código será, pois cada vez mais é gerado um distanciamento do conceito base.
- Esses níveis são chamados de Hierarquia de Classe.
- No exemplo da "classificação biológica", partindo do nível Espécie até Reino, a Hierarquia de Classe teria 6 níveis

Herança

- Veja que, a partir da Espécie *Homo Sapiens*, podemos chegar no Filo *Chordata*, que engloba todos os vertebrados.
- Com isso, a possibilidade de manipular jacarés e pessoas ao mesmo tempo seria possível.
- Entretanto, isto poderia levar a dificuldades de definição do que realmente se deseja modelar
- O fundamento de reuso já explicado é intrinsecamente ligado à herança e também à abstração.
- Quando definimos uma classe da forma mais abstrata possível, é porque necessitamos reusar seu conceito e seus membros em outros conceitos similares.
- A herança deve ser aplicada para isso.

Herança

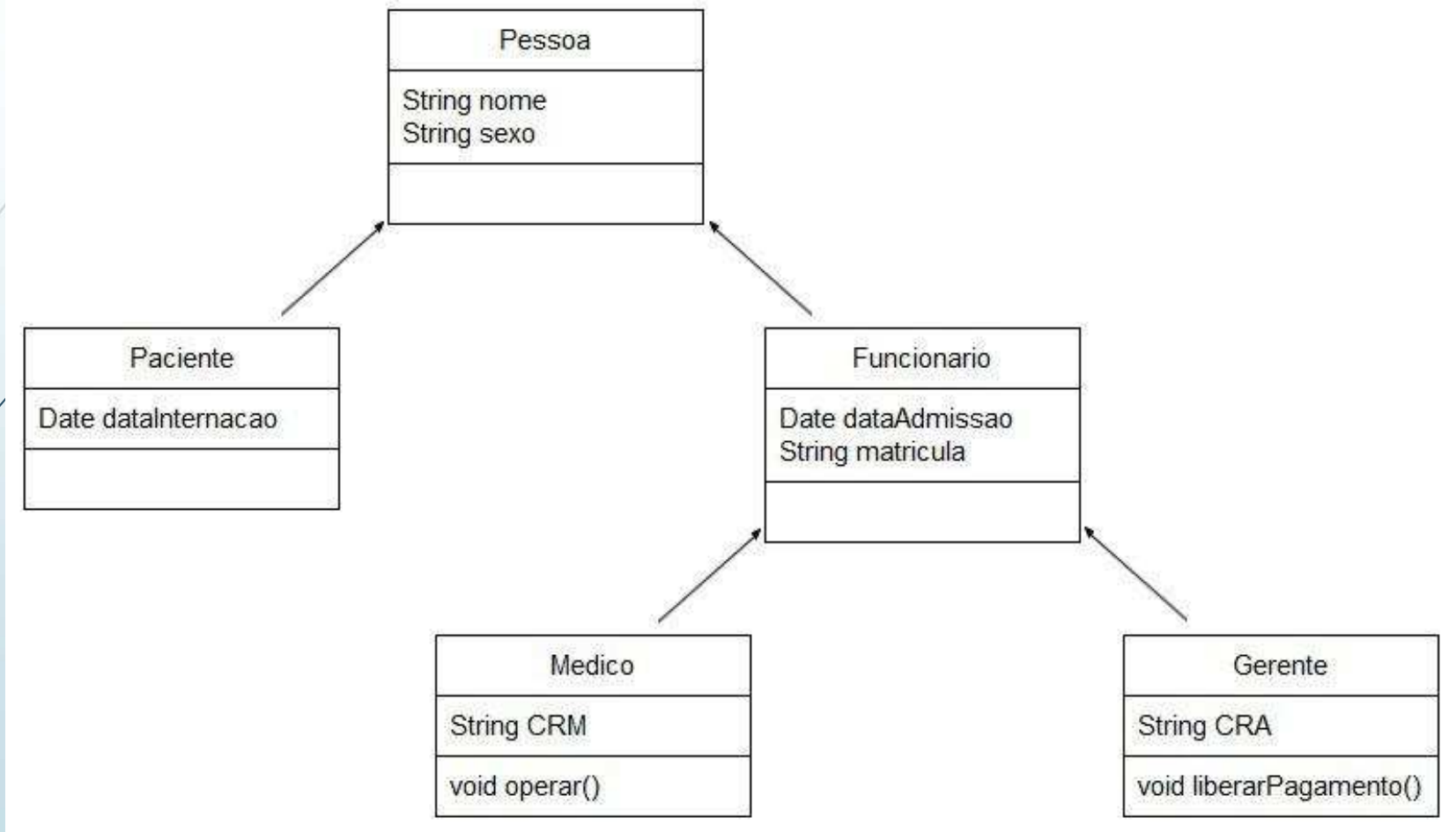
- Quando uma classe herdar de outra, **ela poderá acrescentar novos membros, mas não excluir**.
- Ora, se a ideia é **reusar para evitar repetição**, não teria lógica excluir código.
- Além disto, a grande vantagem da herança é a definição de subtipos.



Herança

- Quando utilizamos a herança, estamos dizendo que um conceito "é do tipo" de outro conceito, e esta possibilidade é vital para representar fielmente o mundo real que está se modelando.
- Por exemplo, no hospital, existem vários tipos de pessoas entre as quais podemos citar: pacientes e funcionários. Estes últimos podem ser Médicos, Enfermeiros, Fisioterapeutas, Gerentes, entre outros.
- Nota-se que Médicos, Enfermeiros, Fisioterapeutas, Gerentes são tipos de funcionários. E paciente e funcionário são do tipo pessoa.
- Ou seja, a partir de um tipo, é possível definir outros tipos. A figura a seguir demonstra essa hierarquia

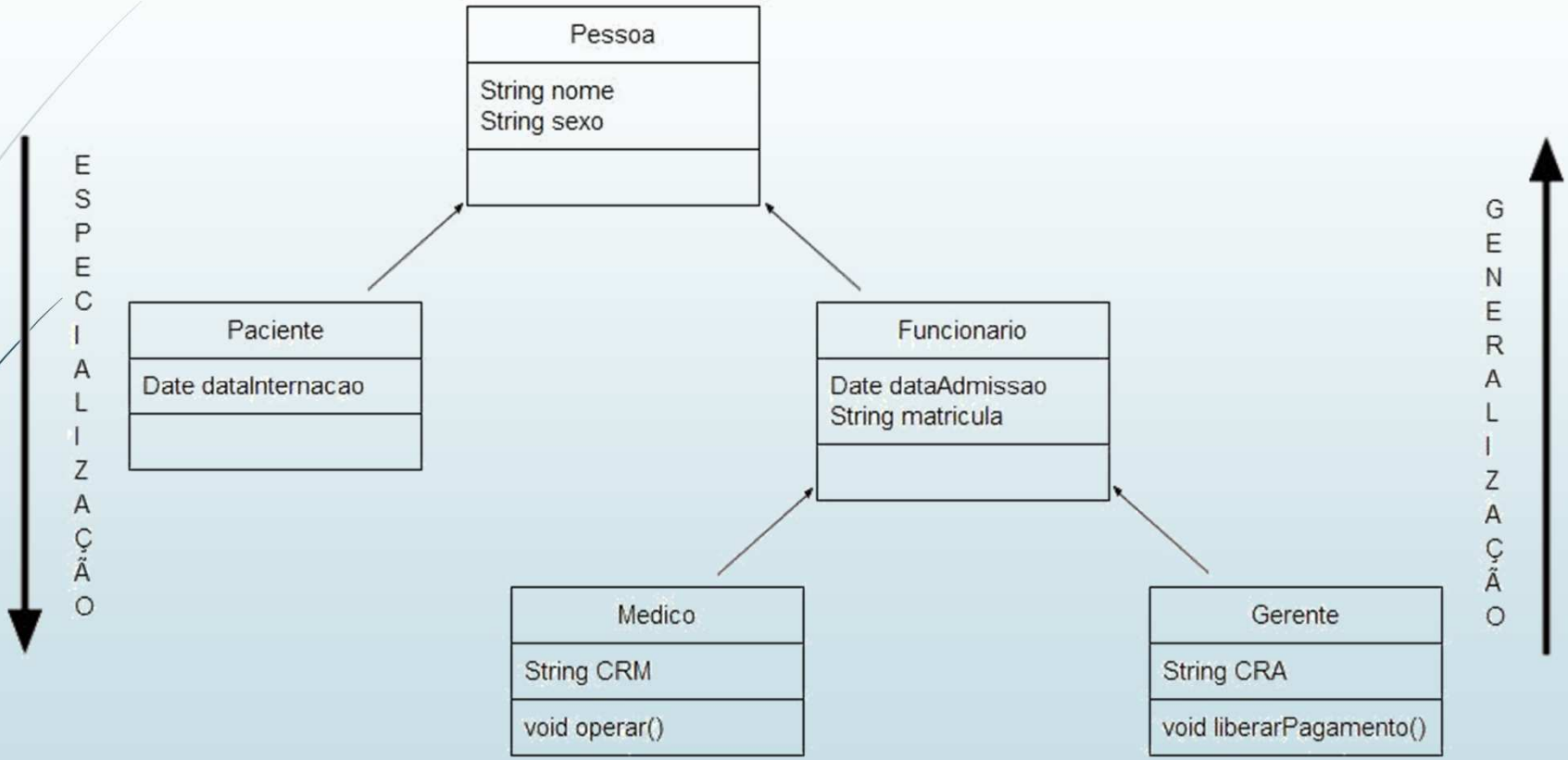
Herança


Pessoa
String nome
String sexo
Paciente
Date dataInternacao
Funcionario
Date dataAdmissao
String matricula
Medico
String CRM
void operar()
Gerente
String CRA
void liberarPagamento()

Herança

- Em cada **subtipo**, membros inerentes a cada um foram sendo acrescentados de acordo com a necessidade, assim cada uma tornou-se completa.
- Em Medico , por exemplo, além de acrescentar seu número de **CRM**, foi acrescentado o método operar() .
- Desta forma, a classe Medico conseguiu definir todos os membros que necessitava em conjunto com a data de admissão, número de matrícula (herdados de Funcionario), nome e sexo (que Funcionario herdou de Pessoa).
- O processo de definir o mais genérico nas classes bases e ir acrescentando nas filhas o mais específico é conhecido como ***Generalização*** e ***Especialização***, respectivamente.
- Quanto mais se sobe na Hierarquia de Classe, mas genérico fica, e quanto mais desce, mais específico.

Herança


Pessoa
String nome
String sexo
Paciente
Date dataInternacao
Funcionario
Date dataAdmissao
String matricula
Medico
String CRM
void operar()
Gerente
String CRA
void liberarPagamento()
ESPECIALIZAÇÃO
GENERALIZAÇÃO

Herança

- Alguns cuidados devem ser tomados quando usamos a herança, como onde colocar os atributos e métodos, e quando realmente devemos usá-la.
- Caso os membros sejam definidos na classe errada, situações esdrúxulas podem ocorrer, pois não representarão a realidade.

Herança

- **Por exemplo, no contexto do hospital, se for levado em consideração que a herança está sendo usada só pelo reuso e não pelos subtipos, seria mais fácil existir somente duas classes filhas de pessoa:** Paciente **e** Funcionario **.**
- **Assim a classe** Pessoa **poderia possuir logo o atributo CRM para conseguir representar um funcionário médico.**
- **Porém, isso levaria a um erro grave, já que pacientes poderiam ter entre seus atributos o CRM.**
- **Pacientes não são médicos! Logo percebemos que pensar em herança só por reúso é um caminho fácil para cometermos erros.**
- **Essa mesma preocupação vale para os métodos.**
- **Por fim, só se pode usar a herança caso a pergunta mágica seja verdadeira: uma coisa é a outra?**

Herança (codificação - Java)

```
//Java
class Pessoa {

  String nome;
  String sexo;

  // get/set e métodos afins
}

class Paciente extends Pessoa {

  Date dataInternacao;

  // get/set e métodos afins
}

class Funcionario extends Pessoa
{

  Date dataAdmissao;
  String matricula;

  // get/set e métodos afins
}
```

```
class Medico extends Funcionario {

  int CRM;

  // get/set e métodos afins

  void operar() {
    // implementação desejada
  }
}

class Gerente extends Funcionario {

  int CRA;

  // get/set e métodos afins

  void liberarPagamento() {
    // implementação desejada
  }
}
```

Herança (codificação – C#)

```
class Pessoa
{
   String nome;
   String sexo;
   // get/set e métodos afins
}

class Paciente : Pessoa
{
   Date dataInternacao;
   // get/set e métodos afins
}

class Funcionario : Pessoa
{
   Date dataAdmissao;
   String matricula;
     // get/set e métodos
   afins
   }
```

```
class Medico : Funcionario
{
   int CRM;
   // get/set e métodos afins
   void Operar()
   {
      // implementação desejada
   }
}

class Gerente : Funcionario
{
   int CRA;
   // get/set e métodos afins
   void LiberarPagamento()
   {
      // implementação desejada
   }
}
```


Herança

- **Em** Java , **a herança é feita com o uso da palavra reservada** extends .
- **Já em** C# **deve ser utilizado o símbolo** : **(dois pontos).**

- **A partir desses códigos, verificamos que a classe** Pessoa **a superclasse de todas as demais classes.**
- **Mas, em alguns momentos, algumas subclasses terminam sendo superclasses de outras, como** Funcionario **herda de** Pessoa , **então** Funcionario **é subclasse da superclasse** Pessoa .
- **Porém,** Medico **herda de** Funcionario , **então neste momento** Funcionario **passou a ser uma superclasse e** Medico **uma subclasse.**
- **Nota-se que basta mudar o ponto de referência que as subclasses e superclasses podem mudar**

Informação

- **O TIPO DE DADO DATE**
- **Tanto** Java **como** C# **proveem um tipo de dado não primitivo para manipulação de datas, chamado** Date **.**
- **Este é uma classe e possui vários métodos que auxiliam na manipulação de datas.**

- **NA HERANÇA, UMA SUBCLASSE TEM ACESSO A TODOS OS MEMBROS DA SUPERCLASSE?**
- **Sim e não. Na verdade, ainda existe um conceito que precisamos explicar para poder responder esta pergunta de forma completa.**
- **Quando vermos o conceito de visibilidade, poderemos responder melhor a esta pergunta.**
- **Por enquanto, vamos dizer que sim. Mas depois explicaremos melhor o "não".**

Tipos de Classes

- Em relação a como identificar as classes na herança, além dos conceitos de superclasse e subclasse, existem ainda dois outros modos de como representar as classes: abstratas e concretas.
- Uma classe abstrata tem como principal função ser a implementação completa do conceito de abstração.
- São classes que representam conceitos tão genéricos que não vale a pena trabalhar com eles diretamente.
- Eles estão incompletos e devem ser completados pelas classes que herdarem dela, ou seja, seus subtipos.
- **Por não valer a pena trabalhar diretamente com elas, elas têm uma característica importante: não podem ser instanciadas.**

Classes Abstratas (introdução)

- **Por serem de uso indireto, geralmente classes abstratas estão no topo da hierarquia de classe.**
- **O exemplo do hospital ilustra bem esta situação.**
- **Utilizar diretamente objetos do tipo** Pessoa **talvez não seja útil, afinal, é muito importante identificar quem é** Paciente , Medico , Funcionario **para este nicho de negócio.**
- **Cada um executará uma tarefa diferente dentro do hospital e deverá ser tratado da forma adequada.**
- **Então, embora inicialmente não se tenha definido a classe** Pessoa **como abstrata, agora é possível fazer isto e assim obrigá-la a manipular somente suas subclasses.**
- **O código a seguir ilustra como fazer isso, por meio do acréscimo da palavra reservada** abstract **.**

Classe Abstrata (implementação)

```
//Java
abstract class Pessoa {

  String nome;
  String sexo;

  // get/set e métodos afins
}


//C#
abstract class Pessoa
{
  String nome;
  String sexo;

  // get/set e métodos afins
}
```

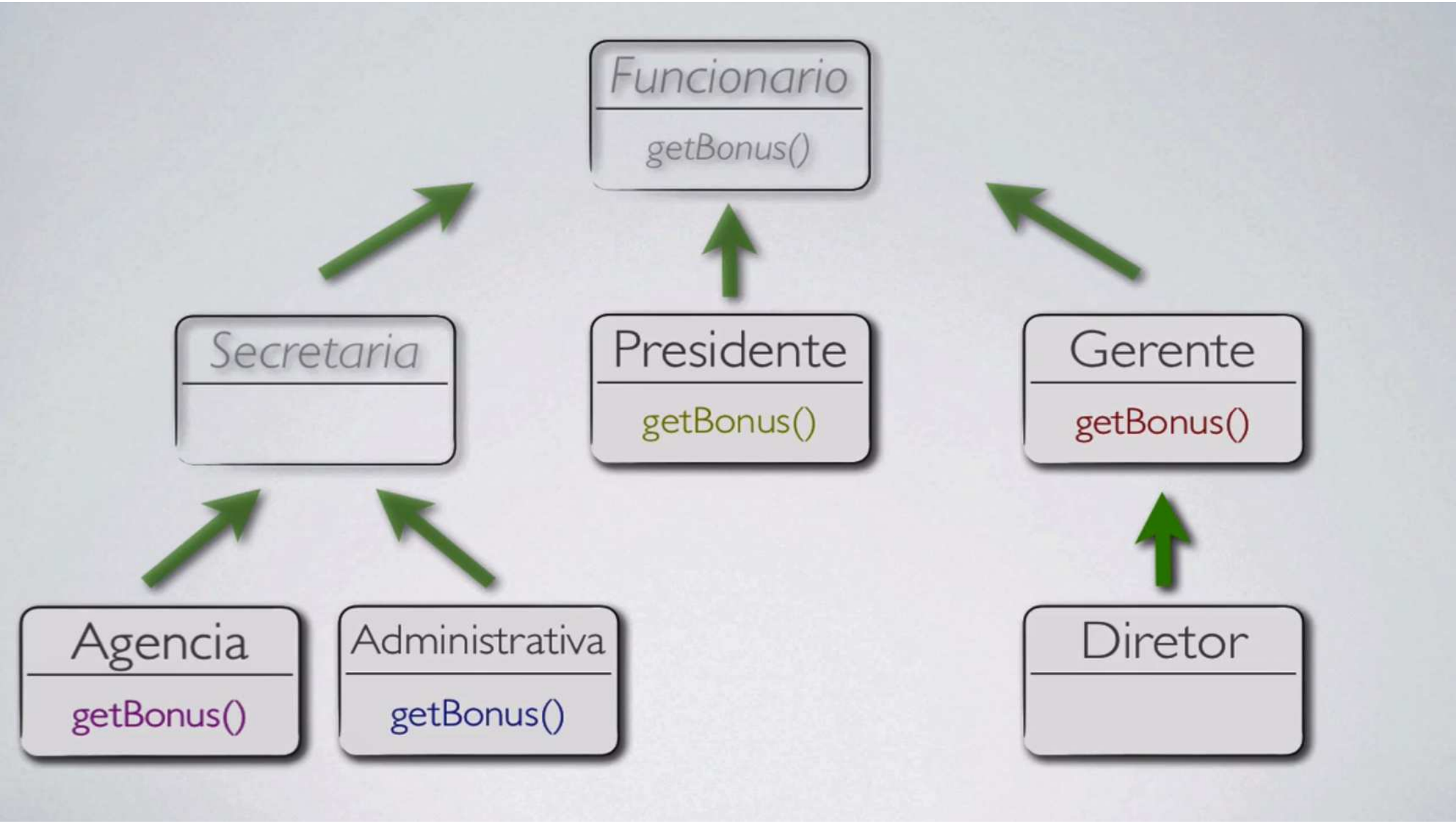
Classe Abstrata

- Tem por objetivo organizar características comuns;
- Não pode ser instanciada;
- Se uma classe tiver pelo menos um método abstrato então ela é abstrata e **DEVE** ser declarada como **abstrata**;

```
public abstract class MinhaClasse
{
     ..............................
     public abstract void facaAlgo();
     ..............................
}
```

Classe abstrata

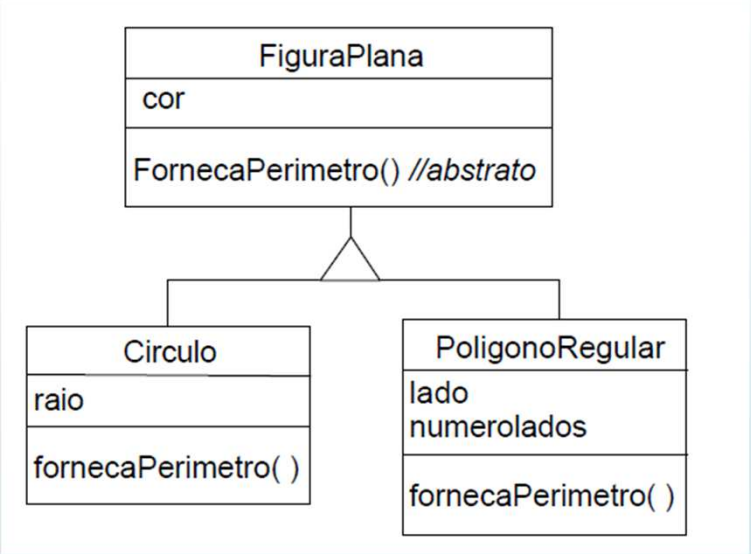
- Se uma subclasse de uma classe abstrata NÃO SOBREPOR os métodos abstratos herdados, então a subclasse também é abstrata e DEVE ser declarada como abstrata.
- Um método abstrato não apresenta corpo.
- Para uma classe abstrata é possível declarar referências desta classe, mas não é permitido instanciar.


Funcionario
getBonus()
Secretaria
Presidente
getBonus()
Gerente
getBonus()
Agencia
getBonus()
Administrativa
getBonus()
Diretor

Polimorfismo

- Poli + morfismo = Polimorfismo --> várias formas
- Ex:

```
umAviao.voe();
umPassaro.voe();
```




```
FiguraPlana figura;
.............................
double perimetro = figura.fornecaPerimetro();
```

/ figura pode conter uma instância da classe Circulo ou PoligonoRegular. Em função da classe da instância referida por figura, será selecionada automaticamente a forma (o método) correta. /**

Polimorfismo

- Polimorfismo --> mesmo tarefa --> formas diferentes em cada classe
  --> métodos diferentes, mas com o mesmo nome. Cada método implementa uma forma.

Interface

- Java não apresenta herança múltipla. Em vez disso, apresenta o conceito de interface.
- Uma interface consiste em uma classe sem atributos de instância e com todos os seus métodos abstratos (inclusive os de classe).
- Uma interface também aceita dados public static final (constantes).

```
public interface FiguraPlana{
     public static final VALORPI = 3.1415;

     public abstract float fornecaArea();

     public abstract float fornecaPerimetro();

}

public interface CompDeslocavel{

     public abstract void deslocarH( float deslocamento);

     public abstract void deslocarV( float deslocamento);

}
```

Interface

```
public class Circulo extends Object implements FiguraPlana, CompDeslocavel
     {
     protected float x;
     protected float y;
     protected float raio;
     public Circulo()
     { x = 0.0f; y=0.0f; raio=0.0f;
}
public Circulo ( float vx, float vy, float vr){
     x = vx; y = vy; raio = vr;
     if (raio < 0 )
          raio = 0;
}
public float informeX(){
     return x;
}
public float informeY(){
     return y;
}
```

Interface

```
public void informeRaio(){
     return raio;
}
public void recebaValorX(float vx){
     x = vx;
}
public void recebaValorY ( float vy ){
     y = vy;
}
public void recebaValorRaio ( float vRaio){
     raio = vRaio;
     if (raio < 0)
          raio=0;
}
public float fornecaArea(){
     // implementação do método abstrato herdado de FiguraPlana
     return VALORPI * Math.pow(raio,2);
}
```

Interface

```
public float fornecaPerimetro(){
    // implementação do método da interface FiguraPlana
    return 2*VALORPI*raio;
}
public void deslocarH( float v){
    // implementação de método da interface CompDeslocavel
    x += v;
}
public void deslocarV ( float v){
        // implementação de outro método da interface CompDeslocavel
        y = y + v;
    }
}
```

Interface

- Em uma interface, também pode-se ter apenas declaração de constantes, como por exemplo:

```
public interface Constantes{

    public static final float SM = 240.00d;

    public static final float VHT = 5.4d;

}
```

Perguntas e Discussão