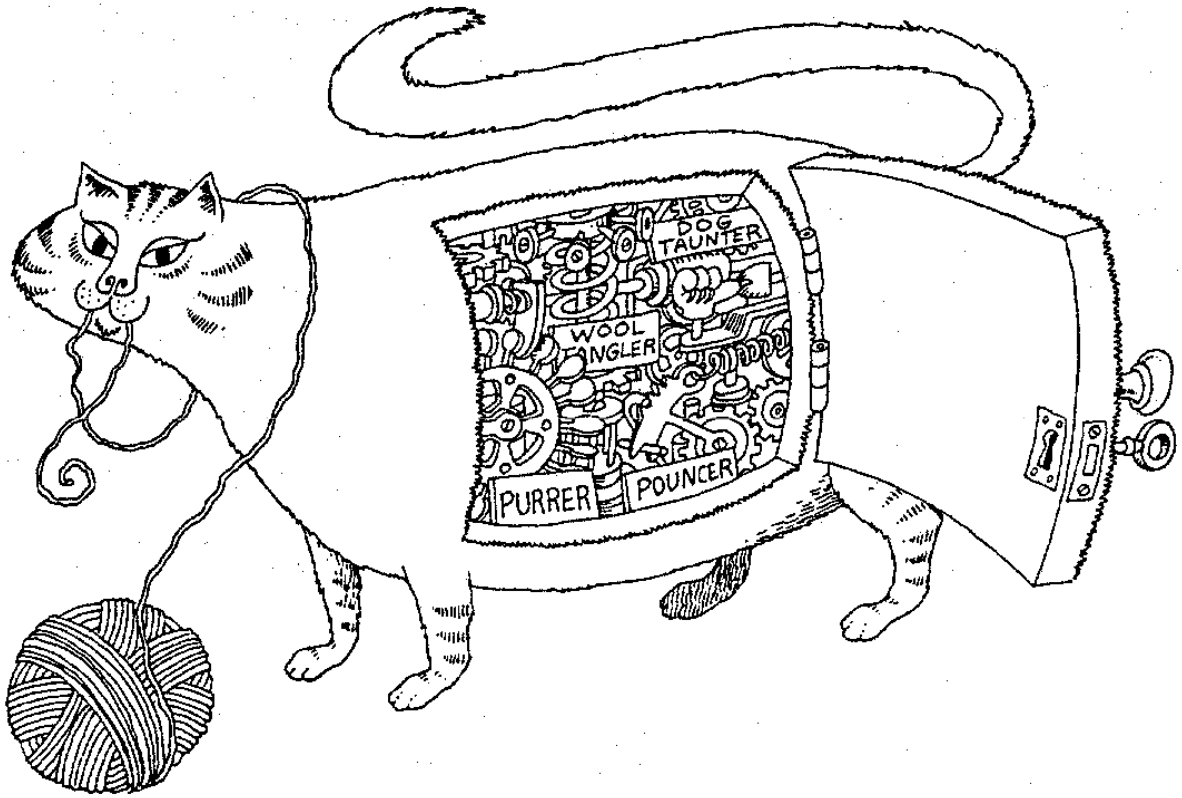
Programação Orientada a Objetos e a Linguagem Java

Rodrigo da Cruz Fujioka
fujiokabr@gmail.com

Objetivos

- Encapsulamento
- Herança

Encapsulamento



Encapsulamento

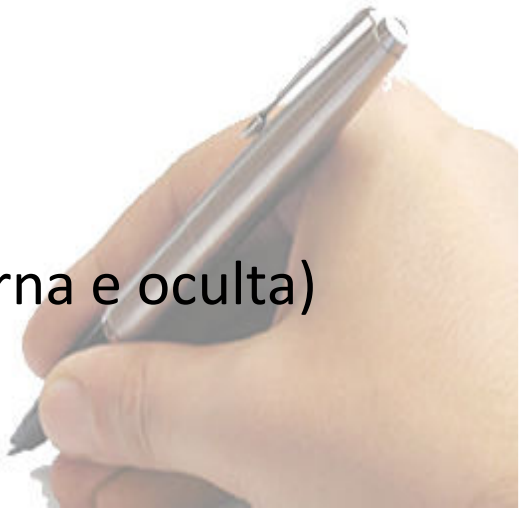
- A abstração de um objeto deve preceder (ter prioridade) à sua implementação
- A implementação, uma vez selecionada, deve ser tratada como um segredo da abstração e, portanto, escondida dos clientes deste objeto
- O encapsulamento serve para proteger a abstração
- "Nenhuma parte de um sistema complexo deve depender dos detalhes internos de outras partes [Booch]"

Encapsulamento

- Conceitos complementares:
  - ✓ A abstração representa um conceito
  - ✓ O encapsulamento, por sua vez, impede os clientes de verem/conhecerem como este conceito foi implementado internamente

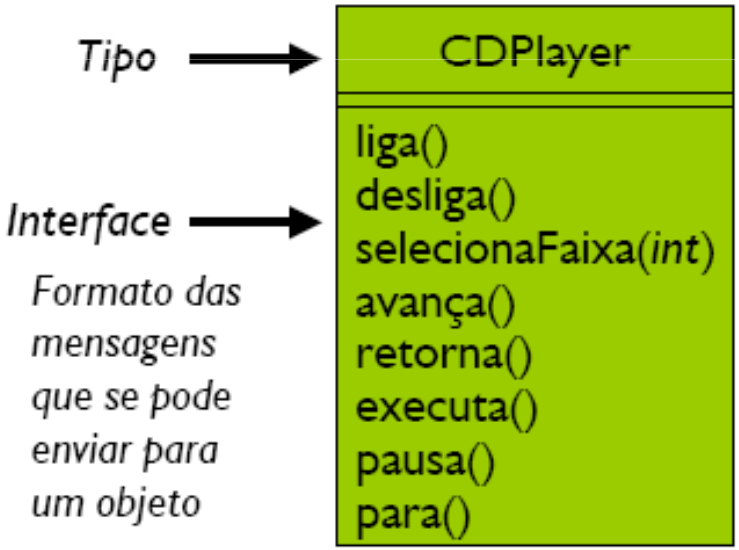
- Para uma abstração realmente funcionar, sua implementação deve estar encapsulada

- Os objetos são formados de duas partes:
  - ✓ Uma interface (visão externa)
  - ✓ Uma implementação (a funcionalidade interna e oculta)



Encapsulamento - Interface

- Através da *interface** é possível utilizá-lo
  - Não é preciso saber dos detalhes da implementação
- O tipo (Classe) de um objeto determina sua interface
  - O tipo determina quais mensagens podem ser enviadas



Em Java

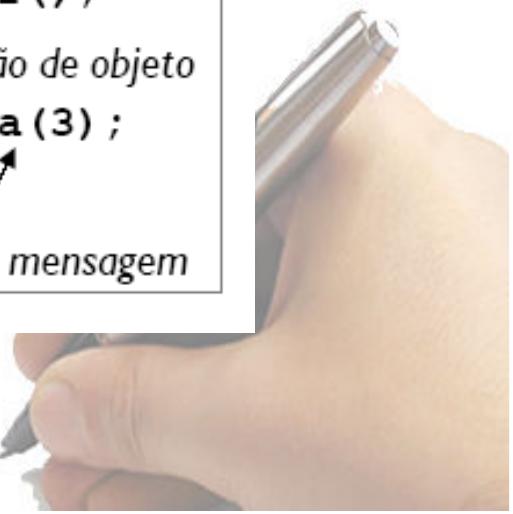
```
(...)
CDPlayer cd1;
cd1 = new CDPlayer();
cd1.liga();
cd1.selecionaFaixa(3);
cd1.executa();
(...)
```

Classe Java (tipo)

Referência

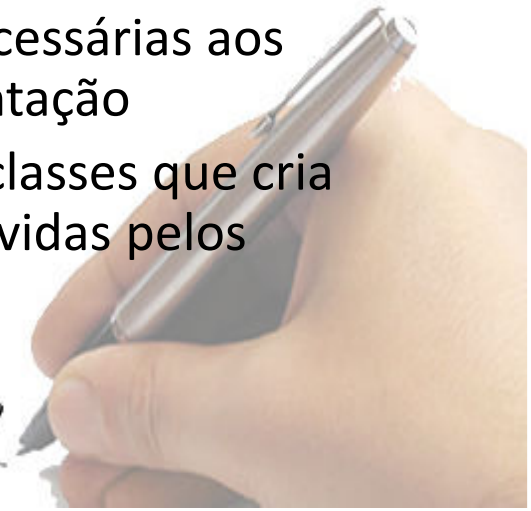
Criação de objeto

Envio de mensagem



Encapsulamento

- A implementação não interessa à quem usa os objetos
- Papel do usuário de classes
  - Não precisa saber como a classe foi escrita, apenas quais seus métodos, quais os parâmetros (quantidade, ordem e tipo) e os valores que são retornados
  - Usa apenas a interface pública da classe
- Papel do desenvolvedor de classes
  - Define novos tipos de dados
  - Expõe, através dos métodos, todas as funções necessárias aos usuários das classes e oculta o resto da implementação
  - Tem a liberdade de mudar a implementação das classes que cria sem que isto comprometa as aplicações desenvolvidas pelos usuários das classes



Encapsulamento

- Definição:
  - "Encapsulamento é o processo de esconder todos os detalhes de implementação de um objeto que não contribuem para suas características essenciais" [Booch]
  - "A habilidade de mudar a representação de uma abstração sem perturbar quaisquer de seus clientes é o principal benefício do encapsulamento" [Booch]
- Lembre-se disto:
  - "O encapsulamento previne acidentes, não fraudes" [Stroustrup]
  - "O encapsulamento não impede o desenvolvedor de fazer coisas estúpidas" [Booch]

Encapsulamento

- Os objetos são formados por:
  - Uma interface: visão externa
  - Uma implementação: funcionalidade interna e oculta (encapsulada)

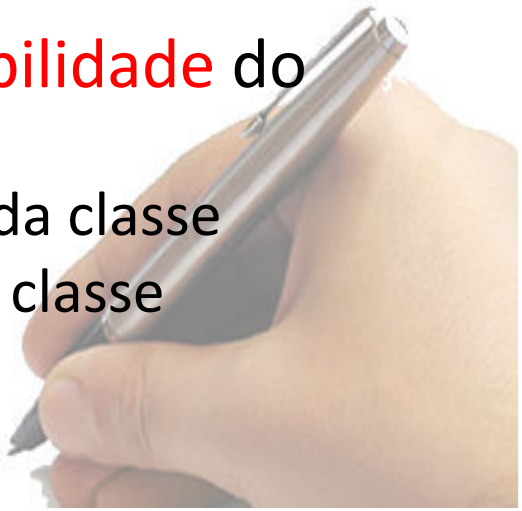
- A implementação não interessa à quem utiliza os objetos (Ex: relógio).

Encapsulamento

- Uma classe Java define um bloco de construção para encapsulamento de propriedades e métodos

- Os métodos de uma classe deveriam acessar apenas as propriedades da própria classe e nunca as de outra classe
  - Promove coesão!

- Modificadores de acesso controlam a visibilidade do encapsulamento de Java
  - `public` tudo que pode ser acessado de fora da classe
  - `private` tudo que é particular e exclusivo da classe



Exemplo de Uma Classe Java

Visibilidade do encapsulamento de Cliente:

```
class Cliente extends Object {

    private String nome;
    private String endereco;
    private char sexo;

    public void setNome( String novoNome ){
        nome = novoNome;
    }
}
```

interno

público

Exemplo de Uma Classe Java

Visibilidade do encapsulamento de Cliente:

```
class Venda {
  private Cliente c;
  ...
  public void mudeCliente( String novo ){
     cli.nome = novo;
  }
}
```

A propriedade `nome` *é privativa da Classe cliente*

Exemplo de Uma Classe Java

Visibilidade do encapsulamento de Cliente:

```
class Venda {

  private Cliente c;

  ...

  public void mudeCliente( String novo ){

      cli.setNome(novo);

  }

}
```

O método `setNome()` é público em cliente

Construtores

- Construtores: utilizados para criar e inicializar novos objetos
  - Em Java os construtores recebem o mesmo nome da classe e não possuem tipo de retorno.
  - Deve-se usar os construtores quando deseja-se atribuir valores aos atributos de um objeto no momento de sua criação.

Construtores

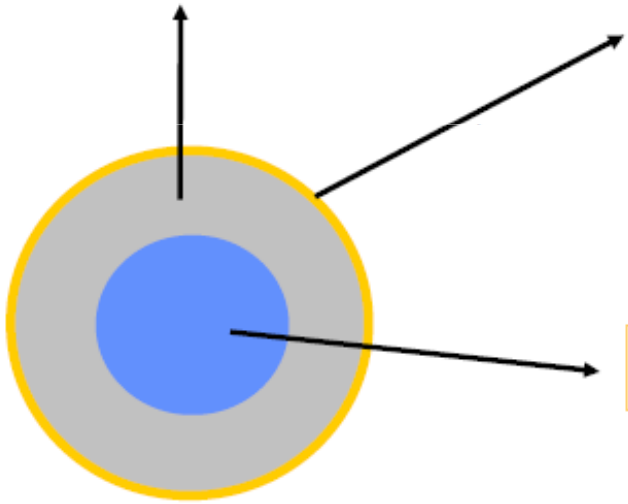

```
public class Cliente {
  private String nome;
  private String endereco;

  public Cliente( String umNome, String umEnd){
     nome = umNome;
     endereco = umEnd;
  }
}
...
Cliente c = new Cliente( "Maria", "Bessa");
```

Encapsulamento

Código dos métodos e construtores

Modifica os dados
Inteligência do objeto



objeto

Interface

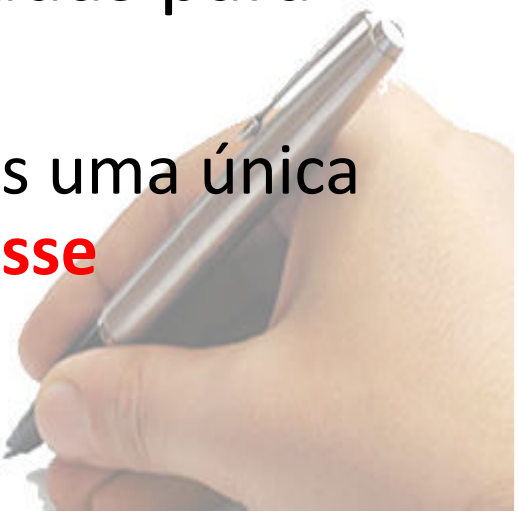
Declara as operações que são públicas (todos podem usar)

Dados (Atributos)

O acesso aos dados é feito através da chamada de um método

Variáveis de Classe

- Uma classe pode ter variáveis contendo informações úteis, como:
  - Número de objetos instanciados pela classe até certo instante
  - Valor médio, mínimo e máximo de uma propriedade
- Essas variáveis não devem ser passadas para os objetos
  - Para cada classe, poderá existir apenas uma única cópia de cada variável: **variável de classe**



Variáveis de Classe

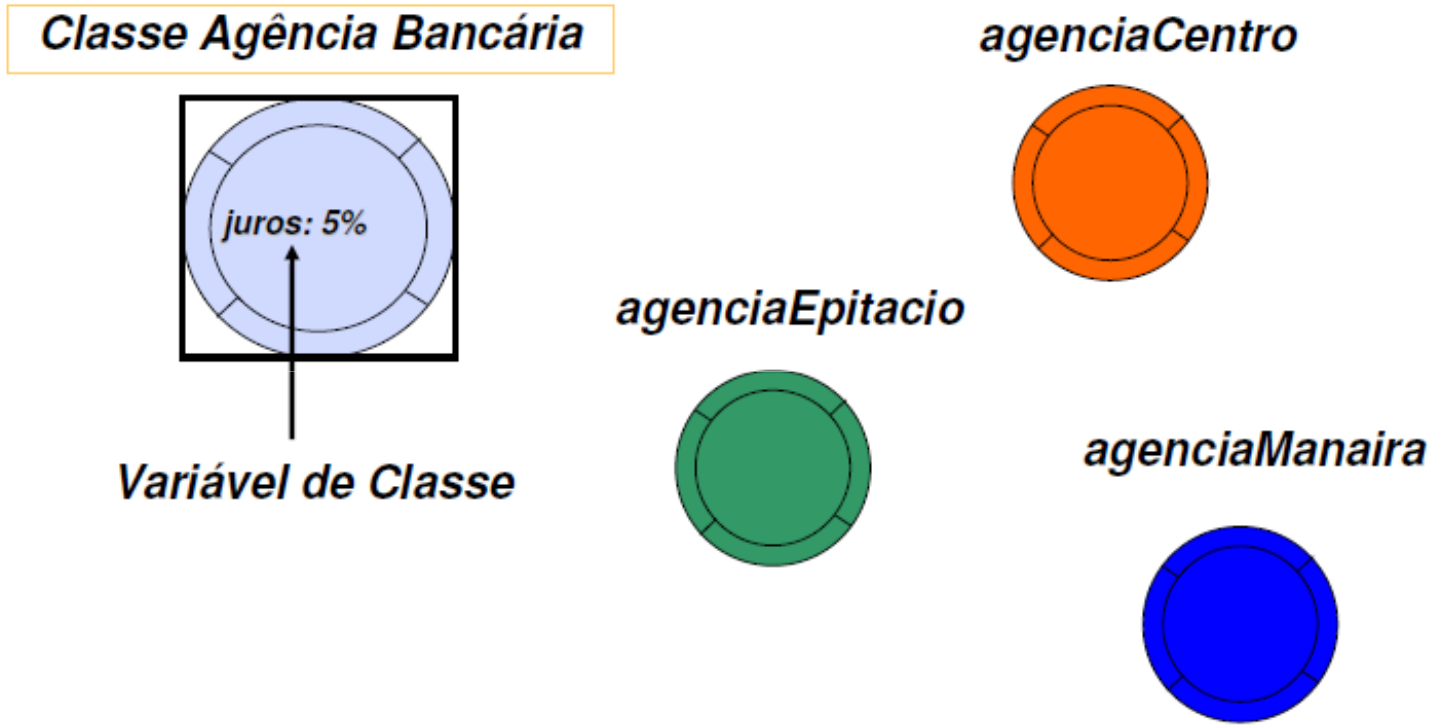
- Uma variável de classe tem sua declaração precedida pela palavra reservada static.
- Todos os objetos de uma classe tem acesso a uma variável de classe e esta é única.
- Variáveis de classe podem armazenar informações como:
  - Número de objetos instanciados pela classe até um certo instante, valor médio de uma propriedade, ...
- A modificação numa variável de classe é percebida por todos os objetos

Variáveis de Classe

- A variável de classe fica armazenada **na classe** e não nas instâncias geradas

- É comum o uso de variáveis de classe para definir constantes
  - PI, MAX_IDADE, MIN_JUROS, ...

Variáveis de Classe



Variáveis de Classe


```
public class MinhaClasse {
  private static int qtdeObjetos = 0; // Variável de Classe
  private int meuNumero;   // Variável de instância
  public MinhaClasse( ) {  // Construtor
   qtdeObjetos++;
        meuNumero = qtdeObjetos;
  }
  public static int getQtdeObjetos() {
   return qtdeObjetos;
  }
}
```

- Forma de acesso: **MinhaClasse.qtdeObjetos**

Variáveis de Classe

- Permitem acessar parte do comportamento de uma classe independentemente de suas instâncias.
  - Método showMessageDialog(...) que monta um diálogo para o usuário da classe JOptionPane.
  - Método sqrt( ) que calcula a raiz quadrada da classe Math.

Variáveis de Classe

- Comportamento é da classe e não de um objeto em particular
  - É como se a classe tivesse operações próprias: **método de classe**

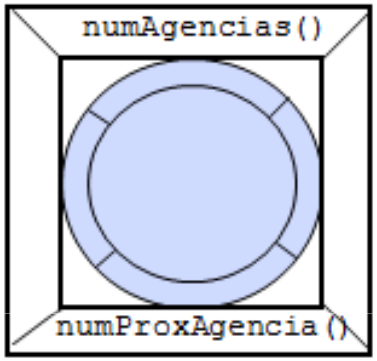
Variáveis de Classe


```
public class MinhaClasse {
  private static int qtdeObjetos = 0; // Variável de Classe
  private int meuNumero;   // Variável de instância
  public MinhaClasse( ) {  // Construtor
   qtdeObjetos++;
        meuNumero = qtdeObjetos;
  }
  public static int getQtdeObjetos() {
   return qtdeObjetos;
  }
}
```

- **Forma de acesso:** `MinhaClasse.getQtdeObjetos()`

Variáveis de Classe

- Exemplos de métodos de classe:



Classe Agência Bancária

- O método **`numAgencias()`** poderia consultar o número de objetos instanciados até o momento

- O método **`numProxAgencia()`** poderia informar o próximo número de uma nova agência

Classe - Resumo

- O corpo de uma classe é formado basicamente:
  - Propriedades dos objetos da classe (variáveis)
  - Métodos que implementam as operações as quais os objetos poderão executar (procedimentos e funções)
  - Construtores que inicializam as propriedades de um objeto

Classe - Resumo

- O corpo de uma classe é formado basicamente:
  - Propriedades dos objetos da classe (variáveis)
  - Métodos que implementam as operações as quais os objetos poderão executar (procedimentos e funções)
  - Construtores que inicializam as propriedades de um objeto

Classe - Resumo

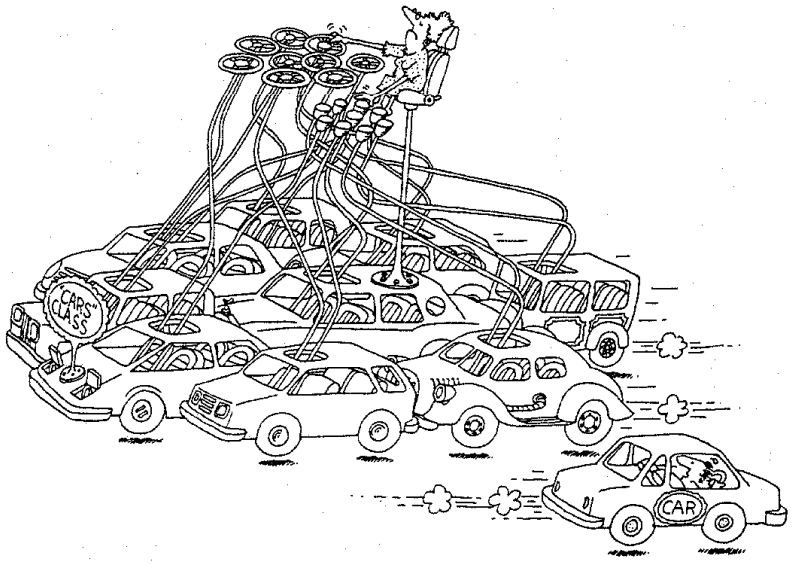
- Uma classe define uma estrutura de dados não-ordenada
  - Pode conter os componentes em qualquer ordem
- Os componentes da classe são seus membros
- Uma classe pode conter três tipos de membros:
  - Membros estáticos ou de classe: não fazem parte do tipo
  - Membros de instância: definem o tipo de um objeto
  - Procedimentos de inicialização

Classe - Resumo

- Membros estáticos ou de classe
  - Podem ser usados através da classe, mesmo quando não há objetos
  - Não se replicam quando novos objetos são criados
- Membros de instância
  - Cada objeto, quando criado, aloca espaço para eles
  - Só podem ser usados através de objetos
- Procedimentos de inicialização
  - Usados para inicializar objetos

Classe - Resumo

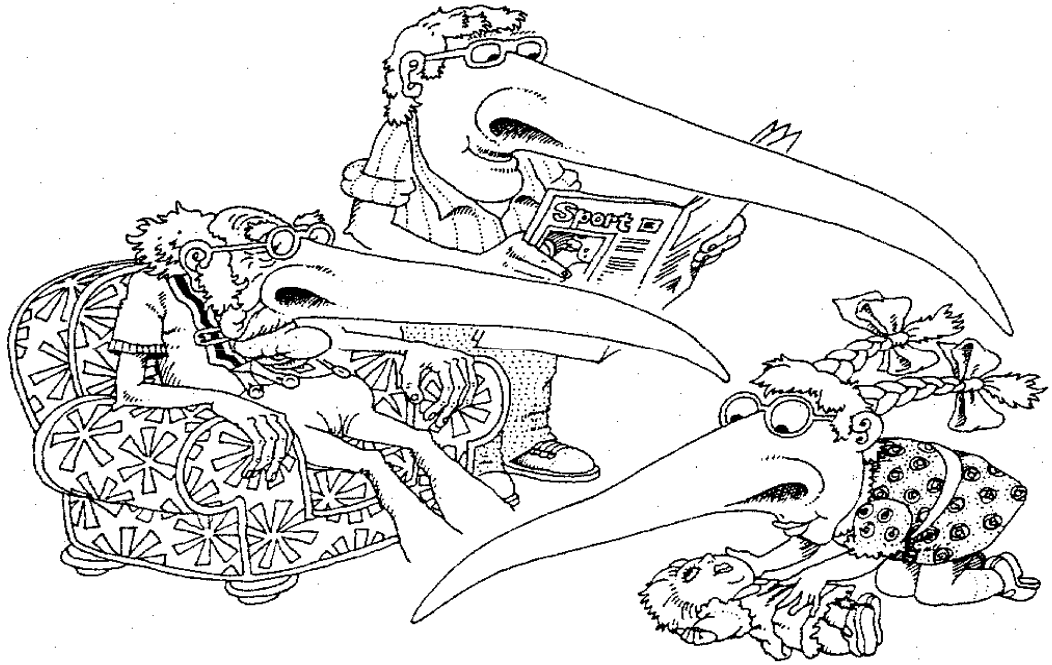
- Uma classe define um conjunto de objetos que compartilham uma **estrutura** e um comportamento comun



Boas Práticas ao Escrever Classes

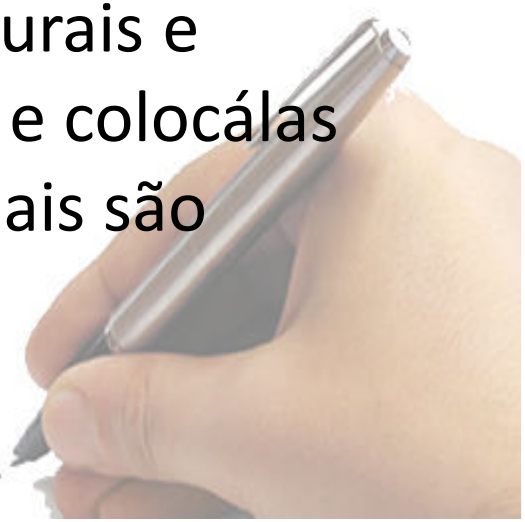
- Use e abuse dos espaços
  - Coloque espaços , deixe organizado, com um tab, ou quatro espaços, os membros de uma classe
- A ordem dos membros não é importante, mas melhora a legibilidade do código
  - Mantenha os membros do mesmo tipo juntos (não misture métodos de classe com métodos de instância)
  - Declare os atributos antes ou depois dos métodos (não misture métodos com construtores ou variáveis)
  - Mantenha os construtores juntos, de preferência, bem no início da classe após os atributos

Herança



Herança

- Ao modelar um conjunto de classes, podemos encontrar classes semelhantes na estrutura e no comportamento.
- Diante disto temos duas soluções:
  - Duplicar código
  - Herança: extrair características estruturais e comportamentais que sejam comuns e colocálas em classes mais gerais a partir das quais são definidas classes mais específicas.

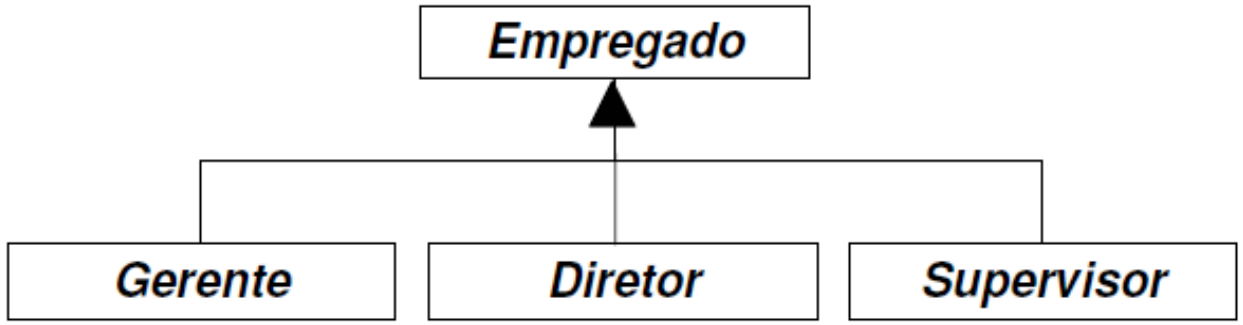


Herança

- A idéia central de herança é que novas classes são criadas a partir de classes já existentes.
  - Subclasse herda de uma Superclasse
  - Subclasse é mais específica que a Superclasse
- Herança é uma técnica para prover suporte a especialização
  - Classes mais abaixo: especializadas
  - Classes mais acima: genéricas

Herança

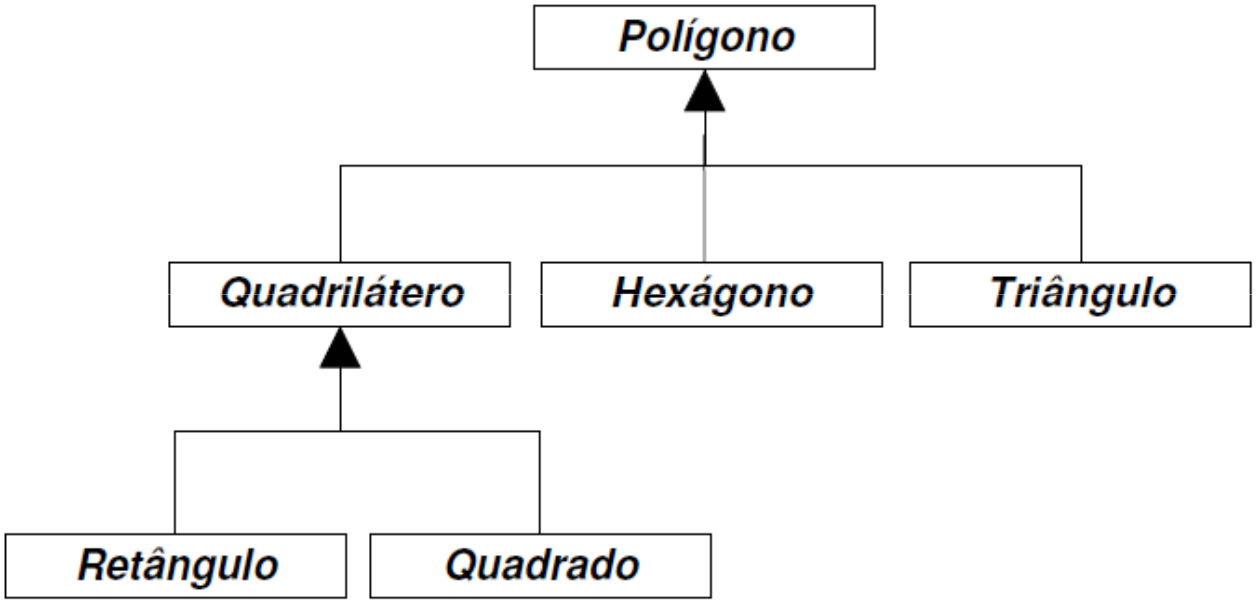
- Métodos e variáveis internas são herdados por todos os objetos dos níveis mais abaixo
- Várias subclasses podem herdar as características de uma superclasse
- A palavra chave extends define de qual classe se desejar herdar



Herança

- Se B é uma subclasse de A, então:
  - Os objetos de B suportam todas as operações suportadas pelos objetos de A, exceto aquelas que foram redefinidas
  - O objetos de B incluem todas as variáveis de instância de B + todas as variáveis de instância de A

Árvore de Herança



Árvore de Herança

- Todos os objetos herdam características definidas em polígono]
- Retângulo e Quadrado são especializações de Quadrilátero
- Em todos os casos, cada subclasse possui uma única superclasse

Benefícios Herança

- Como código pode ser facilmente reutilizado, a quantidade de código a ser adicionado numa classe pode diminuir bastante
  - Subclasses provêem comportamentos especializados tomando como base os elementos comuns criados na superclasse
  - A reutilização pode ser realizada mais de uma vez por outras novas subclasses

Benefícios Herança

- Potencializa a manutenção de sistemas
  - Maior legibilidade do código existente
  - Quantidade menor de código a ser acrescentado

Dúvidas ?

www.rodrigofujioka.com

? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ?

Polimorfismo

? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ?