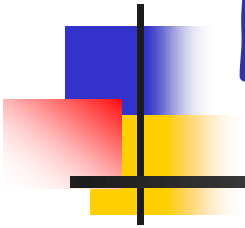
UNIVERSIDADE CATÓLICA DE PERNAMBUCO

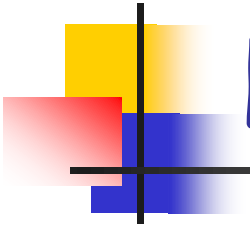


Herança

Universidade Católica de Pernambuco
Ciência da Computação

Prof. Márcio Bueno
poonoite@marciobueno.com

Fonte: Material da Profª Karina Oliveira

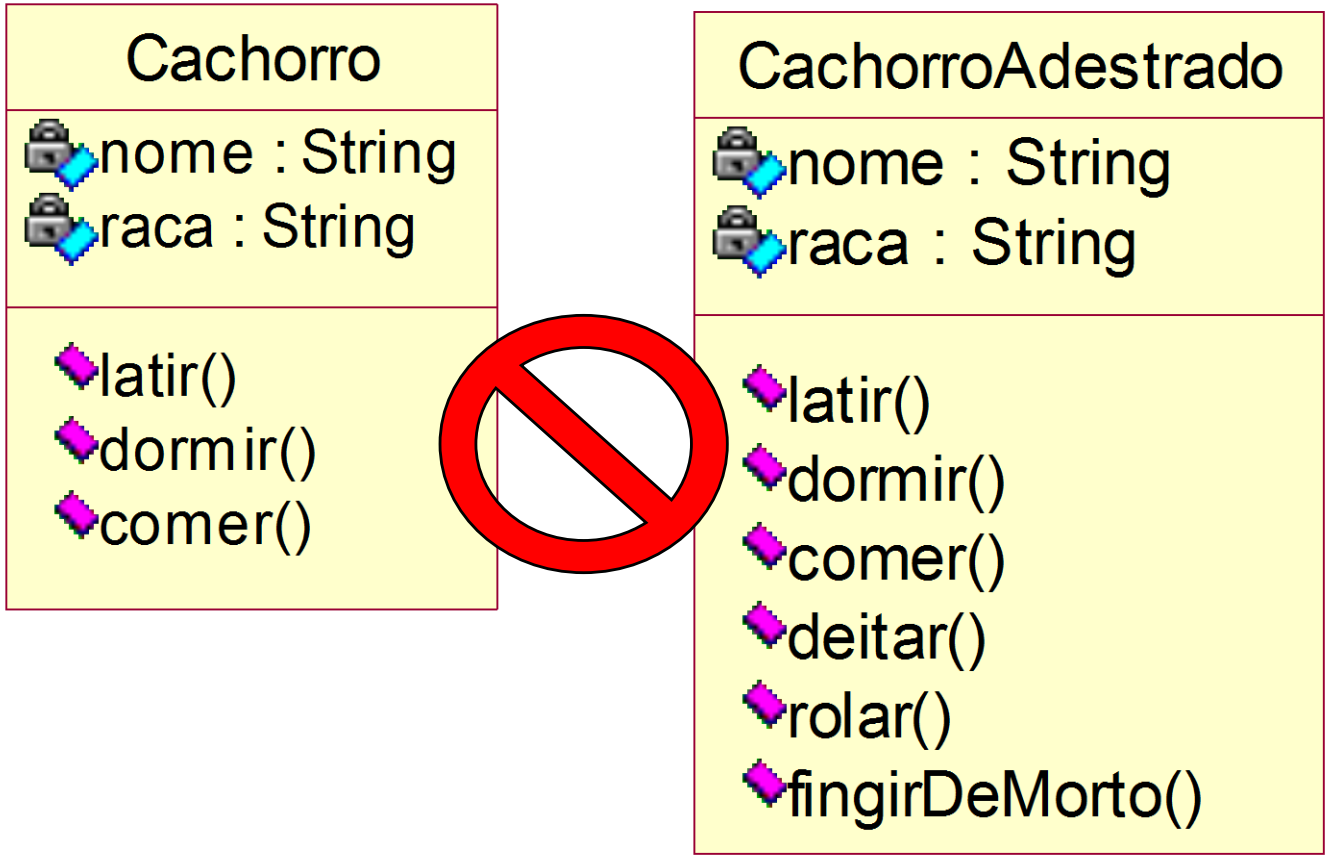


Herança

- Possibilita o **reuso de classes** (código-fonte)
- Usar quando:
  - Desejamos **estender** funcionalidades ou características a partir de um tipo de dado (classe) existente no sistema;
  - Identificamos no sistema vários tipos de dados (classes) com características e funcionalidades **comuns** porém, cada um deles contendo também suas características e funcionalidades **particulares**.

Herança

- Exemplo 1: Acrescentando funcionalidades



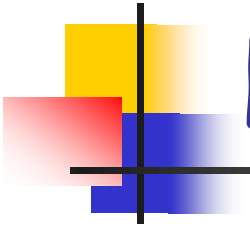
Herança

- Exemplo 2: Acrescentando características

Cliente
nome : String endereco : String

ClientePessoaFisica
nome : String endereco : String cpf : String

ClientePessoaJuridica
nome : String endereco : String cnpj : String

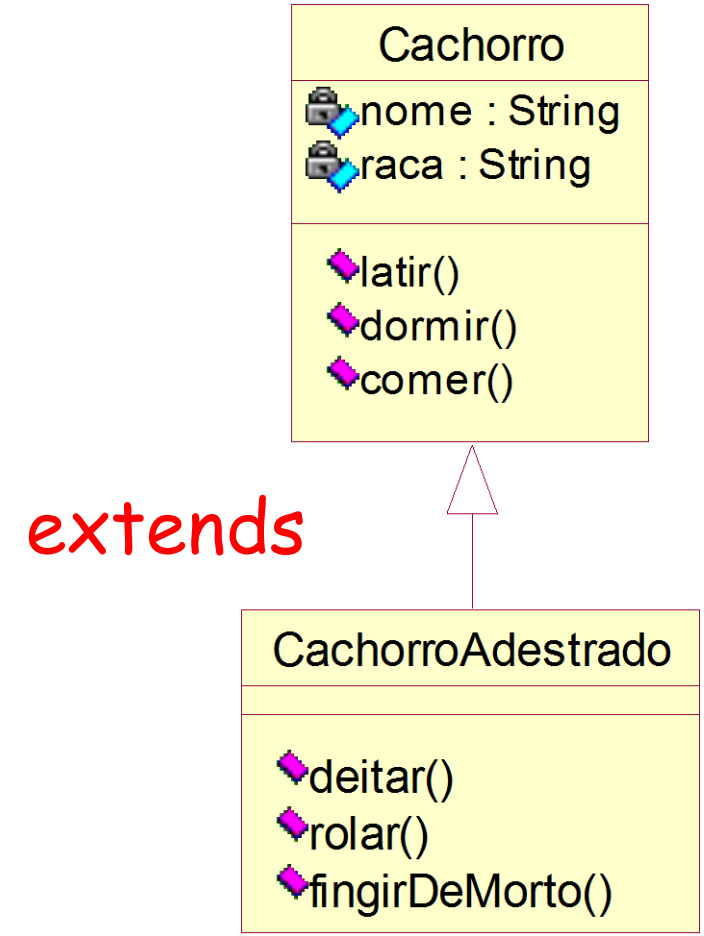


Herança

- Quando dizemos que:
  - Uma classe B **herda** de uma classe A ou
  - Uma classe B **é subtipo de** uma classe A ou
  - Uma classe B **é subclasse** de uma classe A
- Significa dizer que todos os atributos e métodos que foram definidos em A também fazem parte de B.
- A palavra reservada utilizada para expressar o conceito de herança em Java é **extends**.

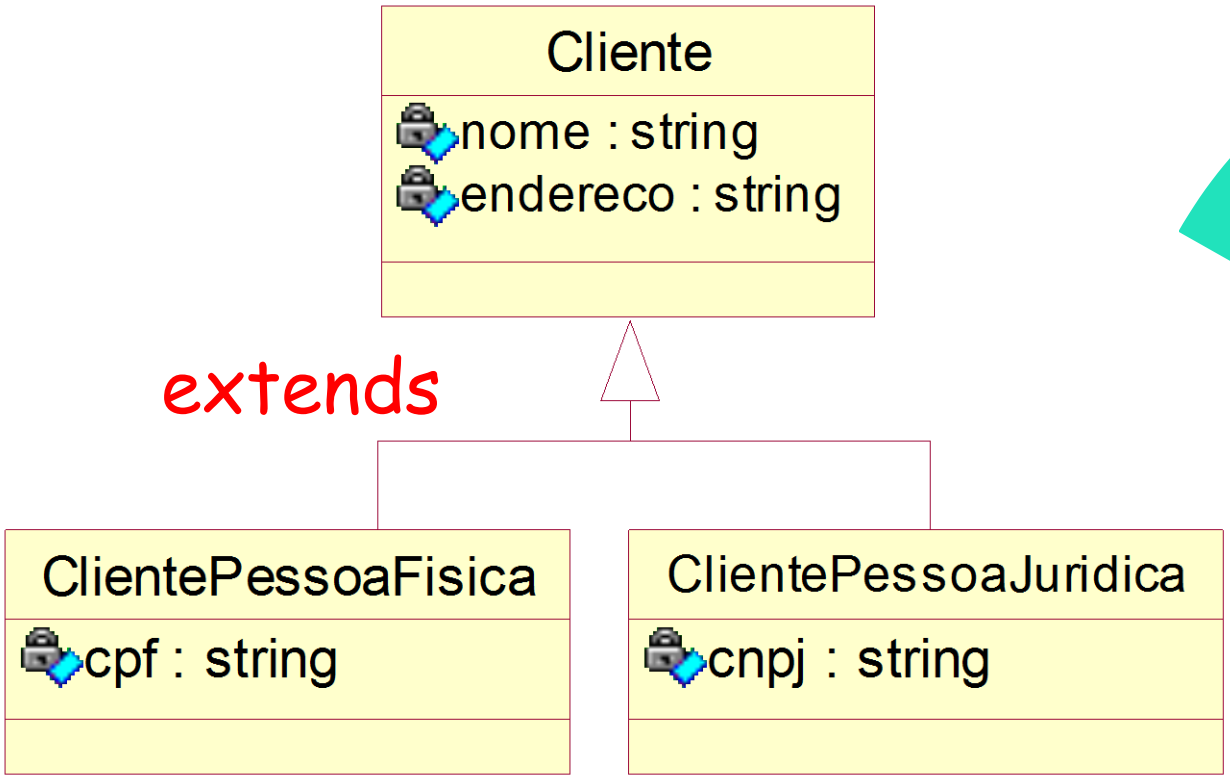
Herança

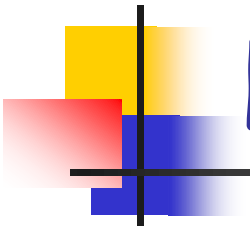
- Exemplo 1: Acrescentando funcionalidades



Herança

- Exemplo 2: Acrescentando características



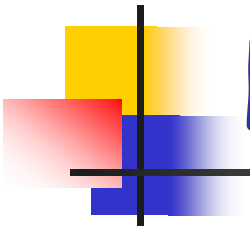


Herança

- Exemplo 2: Código Fonte

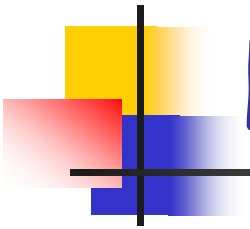
```
public class ClientePessoaFisica extends Cliente {
        // IMPLEMENTAR CORPO DA CLASSE AQUI.
}
```

```
public class ClientePessoaJuridica extends Cliente {
        // IMPLEMENTAR CORPO DA CLASSE AQUI.
}
```



Herança

- Como construir um objeto de uma classe que herda de outra?
  - Encadeamento de construtores
    - Primeiramente, no construtor da subclasse deve-se fazer uma chamada ao construtor da superclasse.
    - Em seguida, faz-se a inicialização dos atributos da própria subclasse.
    - Uso da palavra reservada **super**.
      - Referenciar explicitamente definições (construtores, métodos, atributos) que foram implementadas na superclasse.



Herança

- Exemplo 2: Código Fonte (cont.)

```
public Cliente(String nome, String endereco) {
        this.setNome(nome);
        this.setEndereco(endereco);
}
```

```
public ClientePessoaFisica(String n, String end, String cpf) {
        super(n, end);
        this.setCpf(cpf);
}
```

Herança

- Exemplo 2: Aplicação

```
public class AplicacaoCliente {
  public static void main(String[ ] args) {
    ClientePessoaFisica cli = new
          ClientePessoaFisica("Fulano", "Rua tal", "123");
    AplicacaoCliente.imprime(cli);
  }

  public static void imprime(ClientePessoaFisica c) {
    System.out.println("Nome: " + c.getNome( ));
    System.out.println("Endereço: " + c.getEndereco( ));
    System.out.println("CPF: " + c.getCpf( ));
  }
}
```

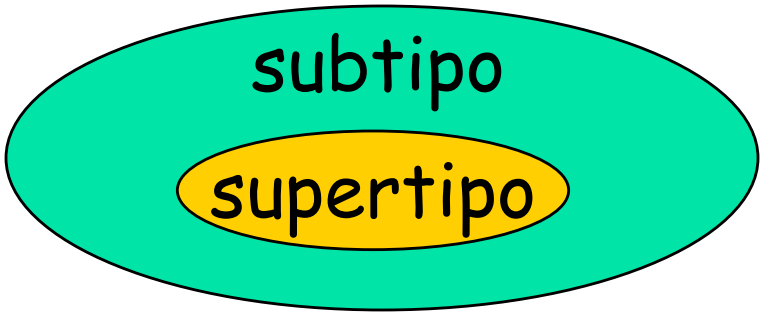
Herança

- Exemplo 2: Aplicação

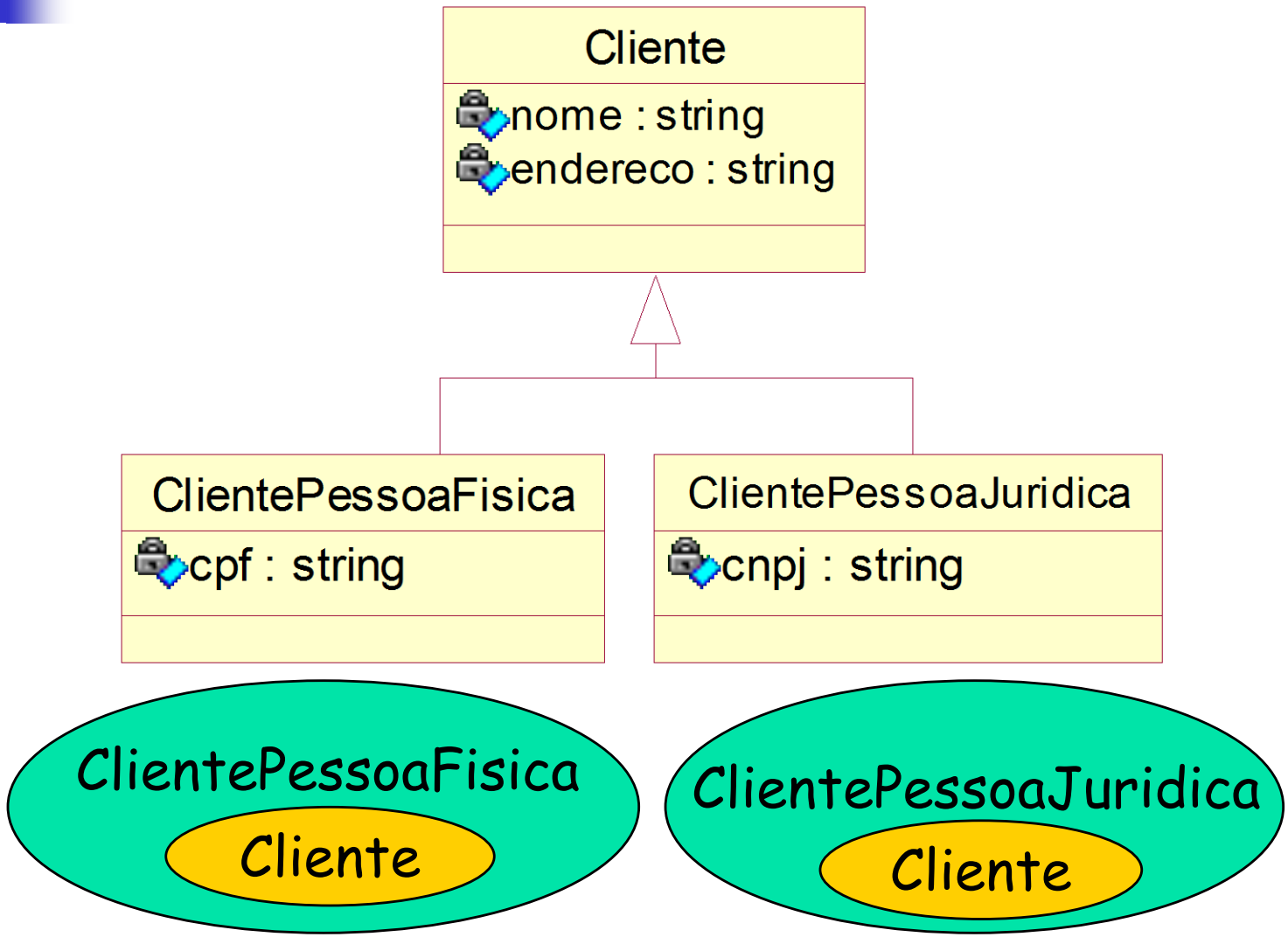
```
public class AplicacaoCliente {
  public static void main(String[ ] args) {
    ClientePessoaJuridica cli = new
          ClientePessoaJuridica("Emp X", "Rua tal", "456");
    AplicacaoCliente.imprime(cli);
  }

  public static void imprime(ClientePessoaFisica c) {
    System.out.println("Nome: " + c.getNome( ));
    System.out.println("Endereço: " + c.getEndereco( ));
    System.out.println("CPF: " + c.getCpf( ));
  }
}
```

Herança e Polimorfismo

- Qualquer elemento, tal como um atributo, variável ou parâmetro de método, que pode referenciar valores de tipos diferentes durante o curso de execução de um programa pode ser considerado como polimórfico.
- Os tipos diferentes que podem ser referenciados por um elemento polimórfico são, exatamente, o supertipo e todos os seus subtipos.



Herança e Polimosfismo



Herança e Polimosfismo

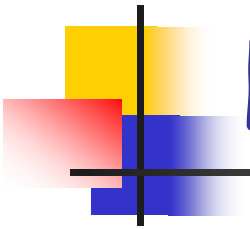
- Voltando ao exemplo 2 – Aplicação:

```
public class AplicacaoCliente {
  public static void main(String[ ] args) {
    ClientePessoaFisica cli = new
          ClientePessoaFisica("Fulano", "Rua tal", "123");
    AplicacaoCliente.imprime(cli);
  }

  public static void imprime (ClientePessoaFisica c) {
    System.out.println("Nome: " + c.getNome( ));
    System.out.println("Endereço: " + c.getEndereco( ));
    System.out.println("CPF: " + c.getCpf( ));
  }
}
```

Herança e Polimosfismo

- Voltando ao exemplo 2 – Aplicação (cont.):

```
public class AplicacaoCliente {
  public static void main(String[ ] args) {
    Cliente cli = new
          ClientePessoaFisica("Fulano", "Rua tal", "123");
    AplicacaoCliente.imprime(cli);
  }

  public static void imprime(Cliente c) {
    System.out.println("Nome: " + c.getNome( ));
    System.out.println("Endereço: " + c.getEndereco( ));
    System.out.println("CPF: " + c.getCpf( ));
  }
}
```



Herança e Polimosfismo

- Problema da Perda de Informação
  - O que acontece é que o compilador toma as suas decisões com base apenas no tipo declarado para a variável.
  - Contudo, Java dispõe de mecanismos que permitem contornar este problema:
    - Conversão de tipos: **cast**
    - O operador **instanceof**
      - Permite determinar qual o tipo de um objeto que está sendo referenciado por uma variável.

Herança e Polimosfismo

- Voltando ao exemplo 2 – Aplicação (cont.):

```
public class AplicacaoCliente {
  public static void main(String[ ] args) {
    Cliente cli = new
          ClientePessoaFisica("Fulano", "Rua tal", "123");
    AplicacaoCliente.imprime(cli);
  }

  public static void imprime (Cliente c) {
    System.out.println("Nome: " + c.getNome( ));
    System.out.println("Endereço: " + c.getEndereco( ));
    System.out.println("CPF: " + c.getCpf( ));
  }
}
```

ERRO DE COMPILAÇÃO!!!

Herança e Polimosfismo

- Voltando ao exemplo 2 – Aplicação (cont.):

```
public class AplicacaoCliente {
  public static void main(String[ ] args) {
    Cliente cli = new ClientePessoaFisica("Fulano", "Rua tal", "123");
    AplicacaoCliente.imprime(cli);
  }
  public static void imprime (Cliente c) {
    System.out.println("Nome: " + c.getNome( ));
    System.out.println("Endereço: " + c.getEndereco( ));
    System.out.println("CPF: " + ((ClientePessoaFisica)c).getCpf( ));
  }
}
```

NÃO CAUSA ERRO DE COMPILAÇÃO.
PORÉM, NÃO EVITA UM EVENTUAL
ERRO EM TEMPO DE EXECUÇÃO!!!

Herança e Polimosfismo

- Voltando ao exemplo 2 – Aplicação (cont.):

```
public class AplicacaoCliente {
  public static void main(String[ ] args) {
    Cliente cli = new ClientePessoaJuridica("Sun", "Rua 2", "456");
    AplicacaoCliente.imprime(cli);
  }

  public static void imprime (Cliente c) {
    System.out.println("Nome: " + c.getNome( ));
    System.out.println("Endereço: " + c.getEndereco( ));
    System.out.println("CPF: " + ((ClientePessoaFisica)c).getCpf( ));
  }
}
```

ERRO EM TEMPO DE EXECUÇÃO!!!

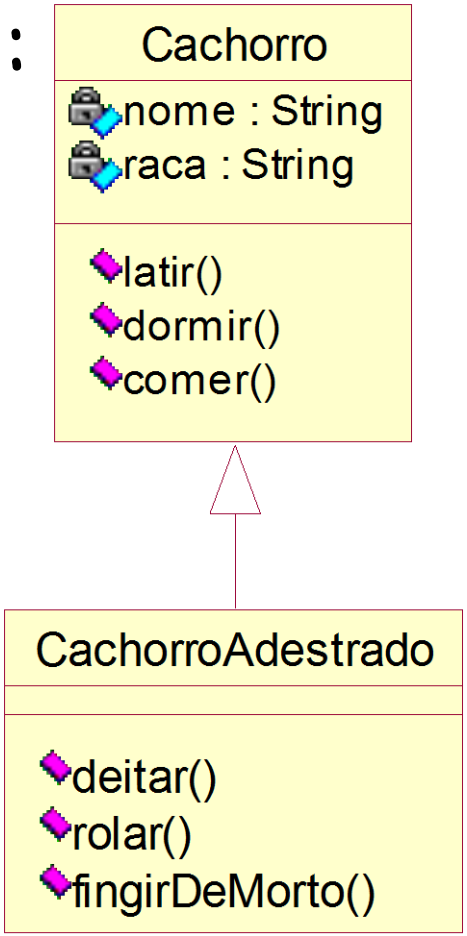
Herança e Polimosfismo

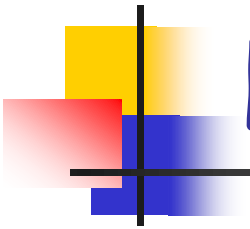
```
public class AplicacaoCliente {
  public static void main(String[ ] args) {
    Cliente cli = new ClientePessoaFisica("Fulano", "Rua tal", "123");
    AplicacaoCliente.imprime(cli);
  }

  public static void imprime (Cliente c) {
    System.out.println("Nome: " + c.getNome( ));
    System.out.println("Endereço: " + c.getEndereco( ));

    if (c instanceof ClientePessoaFisica) {
      System.out.println("CPF: " + ((ClientePessoaFisica)c).getCpf( ));
    } else if (c instanceof ClientePessoaJuridica) {
      System.out.println("CNPJ: " + ((ClientePessoaJuridica)c).getCnpj( ));
    }
  }
}
```

O TESTE DO OPERADOR instanceof GARANTE QUE NÃO OCORRA ERRO EM TEMPO DE EXECUÇÃO!

Herança

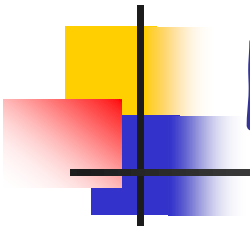
- Exercício 1: Implementar as classes do seguinte modelo:





Herança

- Exercício 1 (Cont.):
  - Atributos:
    - string → não podem receber valores nulos ou string vazia.
  - Métodos de acesso aos atributos (get / set);
  - Um construtor que receba valores para todos os atributos das classes.
  - OBS: Usar a técnica de encapsulamento sugerida no curso.



Herança

- Exercício 1 (Cont.):
  - Implemente um programa chamado Aplicacao que cria dois objetos: um do tipo Cachorro e outro do tipo CachorroAdestrado.
    Ao final, o programa deve imprimir na tela os dados dos objetos criados fazendo chamadas ao método estático imprime implementado na classe Aplicacao:
    - public static void imprime (Cachorro c);