Programação Orientada a Objetos no Java

Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Objetos

O que são Objetos ?
São qualquer coisa na natureza que possua propriedades (características) e comportamentos (operações).

Exemplos de Objetos:

Bolo

Uma partida de futebol

Objeto

Livros

Cachorro

Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Objetos

Orientação a Objetos:
O termo orientação a objetos significa organizar o mundo real como uma coleção de objetos que incorporam ***estrutura de dados e um conjunto de operações*** *que manipulam estes dados.*

Objeto: Pessoa	Propriedades	Comportamentos
	Nome Profissão Data de Nascimento Peso Altura	Andar Correr Trabalhar Chorar Dançar

Objeto: Pássaro	Propriedades	Comportamentos
	Espécie Cor das penas Tamanho Peso	Andar Correr Voar Pousar

Orientação a Objetos

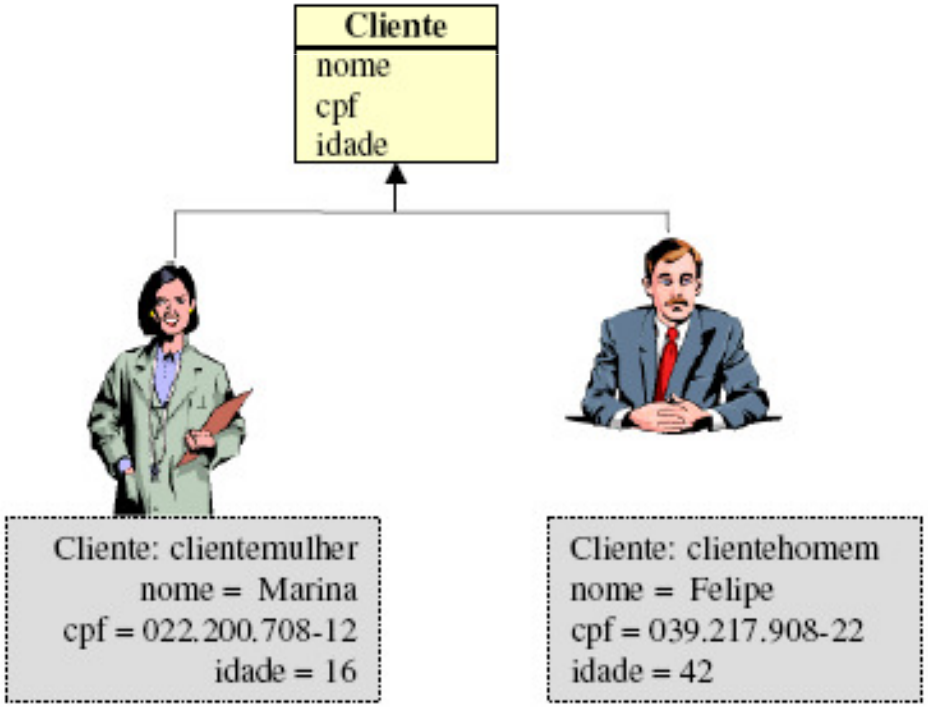
Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Objetos:

Estrutura de um objeto
Um objeto tem identificação, dados e comportamento, atributos e métodos...

Podemos dizer que objeto é espécie da classe, ou seja, uma *instância* da classe



-

Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Objetos:

Resumo

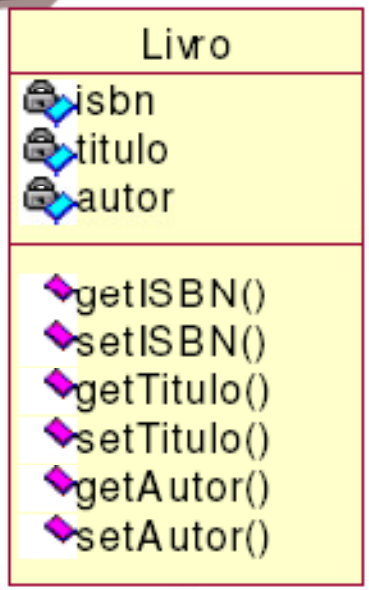
Um objeto possui:

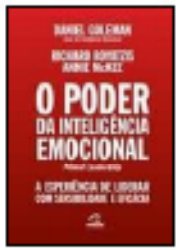
- um estado (definido pelo conjunto de valores dos seus atributos em determinado instante)
- um comportamento (definido pelo conjunto de métodos definido na sua interface)
- uma identidade única

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Classes e Objetos. Exemplo:





ISBN 0747551006
Titulo: O Poder da inteligência Emocional
Autor: Daniel Goleman

ISBN 0747551006
Titulo: Harry Potter and the Order of the Phoenix
Autor: J. K. Rowling

ISBN 8571643512
Titulo: **AS JANELAS DO PARATII**
Autor: Amir Klink

Uma coleção de livros pode ser representada por uma classe chamada Livro.

Cada livro desta coleção é "instância" (objeto) da c ivro.

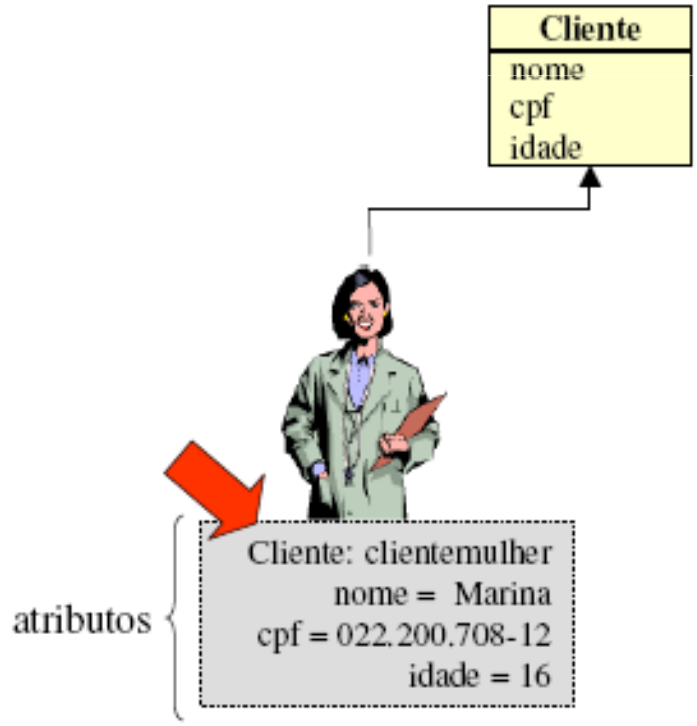
Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Atributos:

- Características presentes nos objetos.
- Valor de todos os atributos = estado do objeto.

- Somente atributos que são de interesse do sistema devem ser descritos na classe.



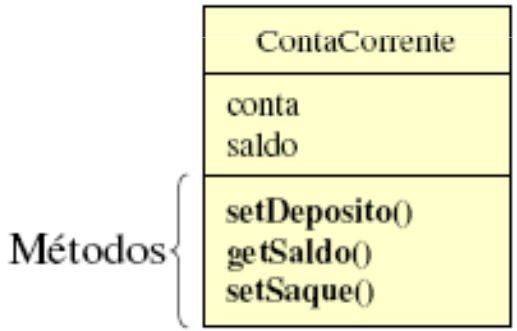
Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Métodos:

Método é a implementação de uma operação. As mensagens identificam os métodos a serem executados no objeto receptor.
Para chamar um método de um objeto é necessário enviar uma mensagem para ele.
Por definição todas as mensagens tem um *tipo* de retorno.



Os métodos encapsulam os atributos

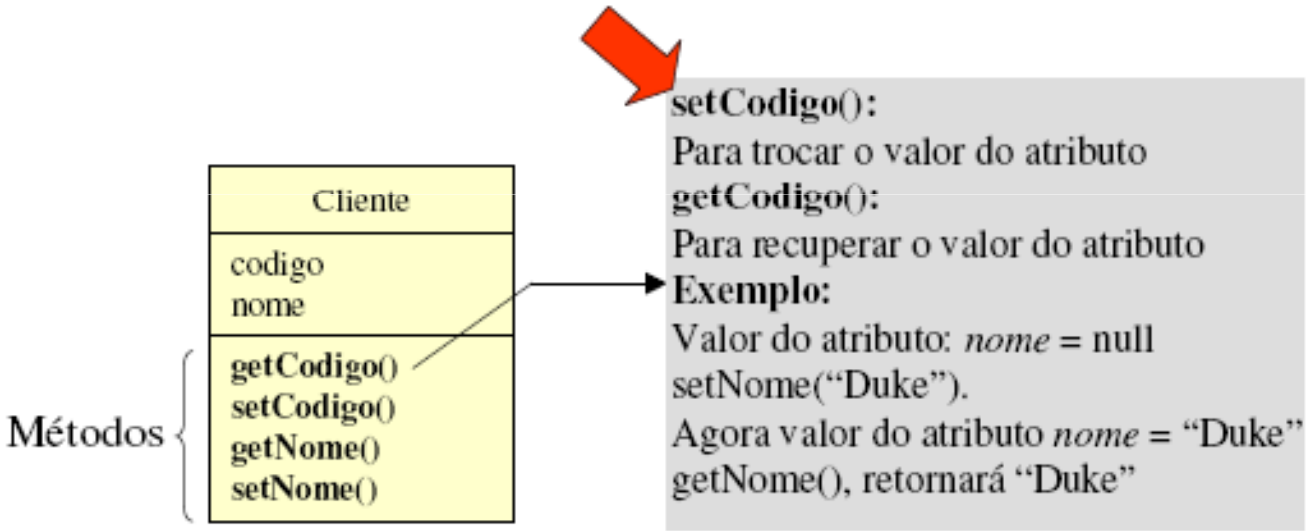
Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Escrevendo os métodos.

Para cada atributo é recomendado escrever um par de métodos, os nomes destes métodos devem começar com setXXXX( ) e getXXX( )



Os métodos encapsulam os atributos

Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Exercício:

Escreve uma classe que melhor represente as figuras dos cartões de créditos.

Nome da Classe
atributos
métodos

Roteiro:
1 - Dê um nome para classe
2 - Identifique quais são as propriedades (atributos) comuns dos cartões
3 - Para cada atributo devo ter uma "par" de métodos get e set.

Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Construindo Classes em Java: Atributos ou propriedades

```
public class Pessoa
{
        public int idade;
        private String nome;
        protected float salario;
}
```

public: pode ser acessado mesmo de outra classe
private: só pode ser acessado da própria classe
protected: acessado pela própria classe e por classes derivadas

Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Construindo Classes em Java: Atributos ou propriedades

```
public class Pessoa
{
        private int idade;
        private String nome;
        protected float salario;

        public void setIdade(int nova_idade) { }
        public int getIdade( ) { }

        public void setNome(String novo_nome) { }
        public String getNome( ) { }

        public void setSalario(float novo_salario) { }
        public float getSalario( ) { }
}
```

Os métodos "set" precisam receber como parâmetro o valor a atribuir os atributos, mas não retornam valor

Os métodos "get" não recebem parâmetros mas retornam o conteúdo do atributo.

Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Construindo Classes em Java: Métodos

```
public void setIdade(int nova_idade)
{
      idade = nova_idade;
}
 public int getIdade( )
{
      return idade;
 }
```

Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Instanciando Objetos

```
public class TesteObjetos
{
      public static void main(String args[])
      {
            Pessoa maria = new Pessoa( );
            Pessoa joao = new Pessoa( );

            maria.setIdade(30);
            joao.setIdade(45);

            System.out.println("Maria=" + maria.getIdade( ) );
            System.out.println("João=" + joao.getIdade( ) );
      }
}
```

Instancia 2 objetos da classe Pessoa

Sobrecarga de Métodos

Métodos com o mesmo nome?

- *Podemos criar vários métodos com o mesmo nome em uma classe, visto que as ações são parecidas mas os **parâmetros** necessários devem ser diferentes.*
- ***Sobrecarregar** um método é criar mais métodos com o mesmo nome mas com parâmetros diferentes.*
- *Os parâmetros podem se diferenciar em tipo e/ou em quantidade, de modo a alterar a **assinatura** do método.*

EXEMPLO DE SOBRECARGA DE MÉTODOS

```
import java.io.*;
class Relogio
{
   private int hora,minuto,segundo;

   public void AcertarRelogio(int h,int m,int s)  {
      acertaHora(h,m,s);
   }

   public void AcertarRelogio(int h,int m)  {
      acertaHora(h,m,0);
   }

   public void AcertarRelogio(int h)  {
      acertaHora(h,0,0);
   }

   public void acertaHora(int h,int m,int s) {
      hora = h;
      minuto = m;
      segundo = s;
   }
  public static void main( String[] args) {
       Relogio pulso = new Relogio();
       pulso.AcertarRelogio(22);
       pulso.AcertarRelogio(22,10);
       pulso.AcertarRelogio(22,10,30);
  }
```

Método AcertarRelogio que recebe TRÊS parâmetros

Método AcertarRelogio que recebe DOIS parâmetros

Método AcertarRelogio que recebe UM parâmetro

Chamada aos métodos sobrecarregados. A Escolha do método que será processado depende dos parâmetros informados.

Orientação a Objetos

Construtores

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Construtores

Construtores são métodos que são invocados automaticamente quando um objeto é instanciado.

São métodos que nunca retornam nada e não possuem tipo

Os construtores sempre tem o mesmo nome da classe a que se refere.

Construtores

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Construtores

```
public class Ponto2D
{
        private int x;
        private int y;

        public Ponto2D()
        {
                x = 0;
                y = 0;
        }

        ....
}
```

O contrutor é chamado automaticamente e zerando os valores de x e y.

Note que esse método não tem tipo de retorno (nem o void)

Construtores

Construtores

```
public class TestePonto2D
{
        public static void main(String args[])
        {
                Ponto2D P1 = new Ponto2D();
        }
}
```

O valor de x e y já está zerado, pois o construtor da classe foi chamado automaticamente.

Sobrecarga de Construtores

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Sobrecarga de Construtores

Como construtores também são métodos, podemos sobrecarregá-los;

Usando o mesmo princípio, terão o mesmo nome e parâmetros diferentes;

Os parâmetros são enviados ao construir o objeto

Orientação a Objetos

Sobrecarga de Construtores

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Sobrecarga de Construtores

```
public class Ponto2D
{
        private int x;
        private int y;

        public Ponto2D()
        {
                x = 0;
                y = 0;
        }

        public Ponto2D(int n_x, int n_y)
        {
                x = n_x;
                y = n_y;
        }
}
```

Construtor sem parâmetros

Construtor com parâmetros

Orientação a Objetos

Sobrecarga de Construtores

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Sobrecarga de Construtores

```
public class TestePonto2D
{
        public static void main(String args[])
        {
                Ponto2D P1 = new Ponto2D();
                Ponto2D P2 = new Ponto2D(120,80);

                P1.setX(100);
                P1.setY(60);

                P1.print();
                P2.print();
        }
}
```

Construtor sem parâmetros

Construtor com parâmetros

Orientação a Objetos

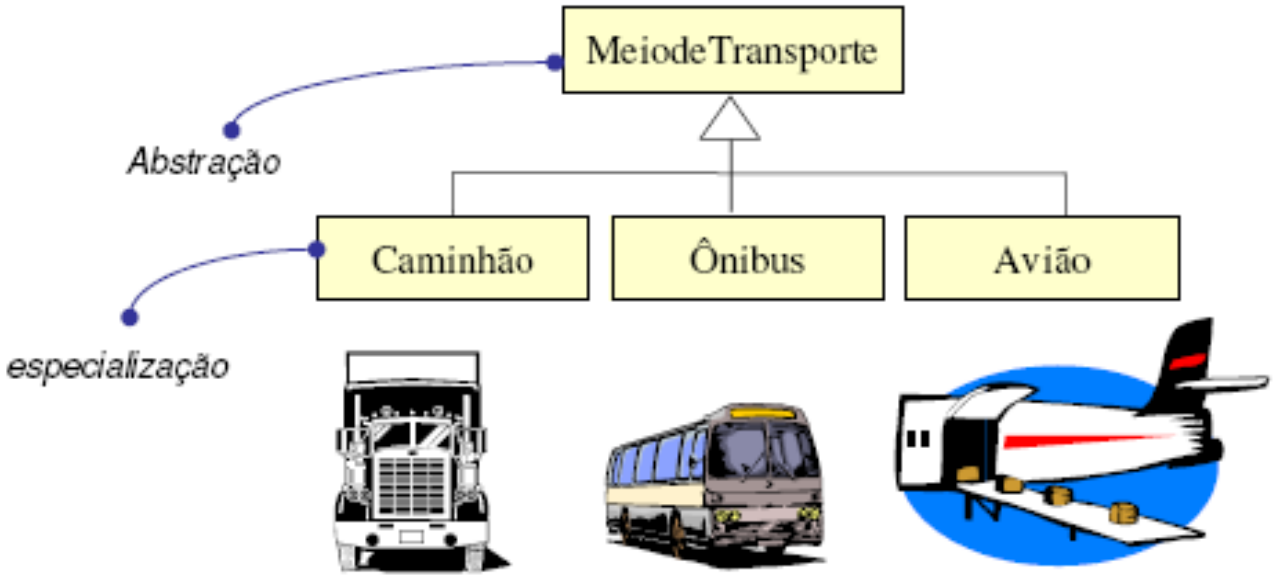
Abstração de Dados

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Abstração de Dados:

O que é abstração?
Podemos dizer abstração é generalização.
Qual é a função da abstração ?
A função da abstração é capturar as propriedades e os comportamentos essenciais, como se fosse uma fatoração, desta forma determina-se o que é importante e o que não é.

Exemplo



Orientação a Objetos

Abstração de Dados

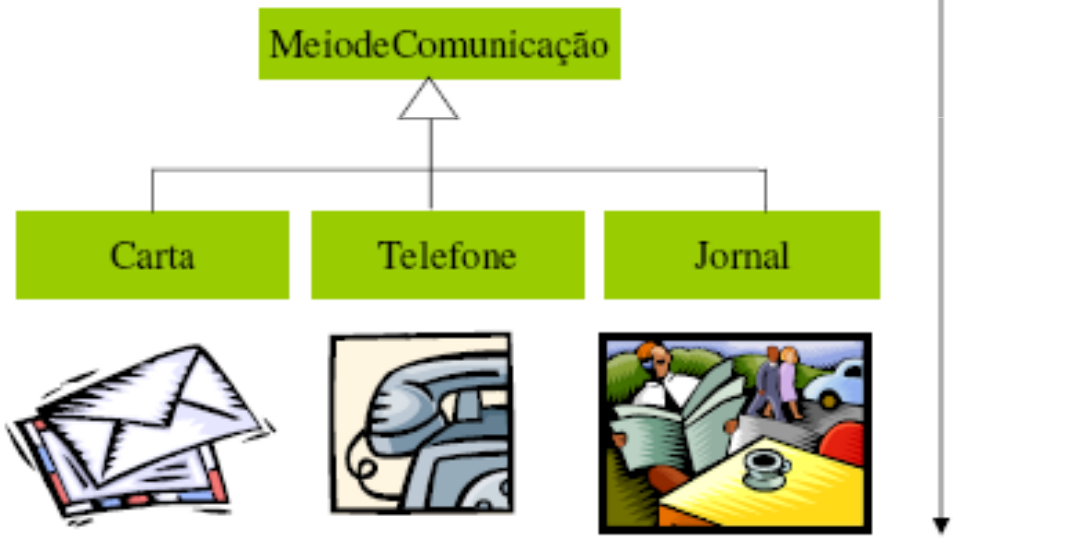
Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Abstração de Dados:

Exemplo

- Abstração: nos ajuda a lidar com a complexidade.

Generalização



Especialização

As classes MeiodeTransporte e *MeiodeComunicação* neste caso são abstratas e ambas podem representar um domínio.

Orientação a Objetos

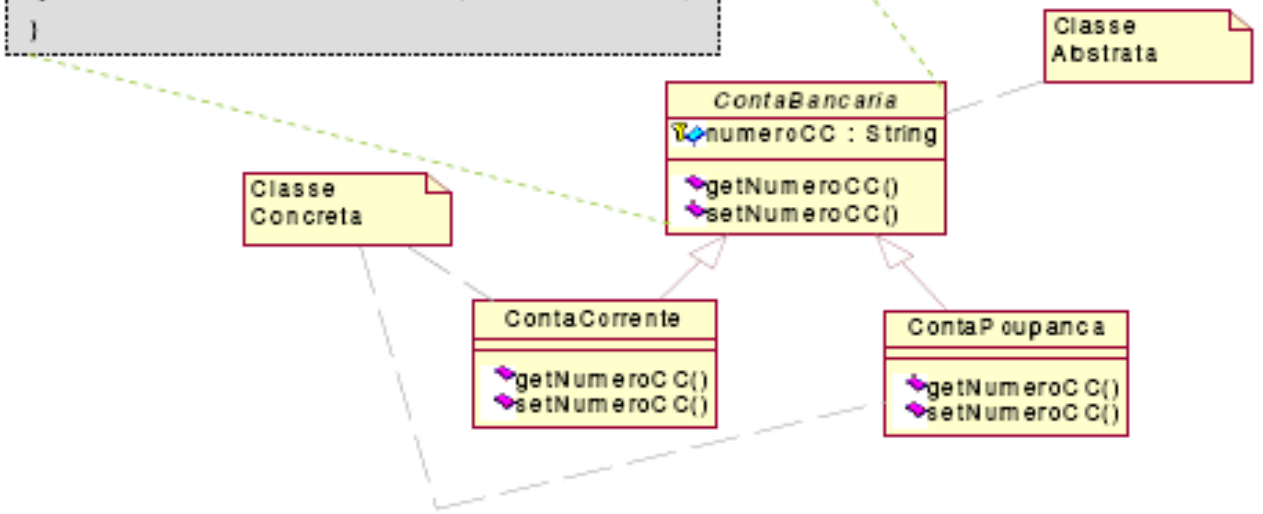
Abstração de Dados

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Abstração de Dados:

Uma classe abstrata é uma classe que:
- Provê organização
- Não possui "instâncias"
- Possui uma ou mais operações (métodos) abstratas

```
public abstract class ContaBancaria {
  public ContaBancaria() {  }
  protected int numerocontacorrente;
  public abstract int getNumeroContaCorrente();
  public abstract void setNumeroContaCorrente(int numerocontacorrente);
}
```



Orientação a Objetos

Herança

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Herança

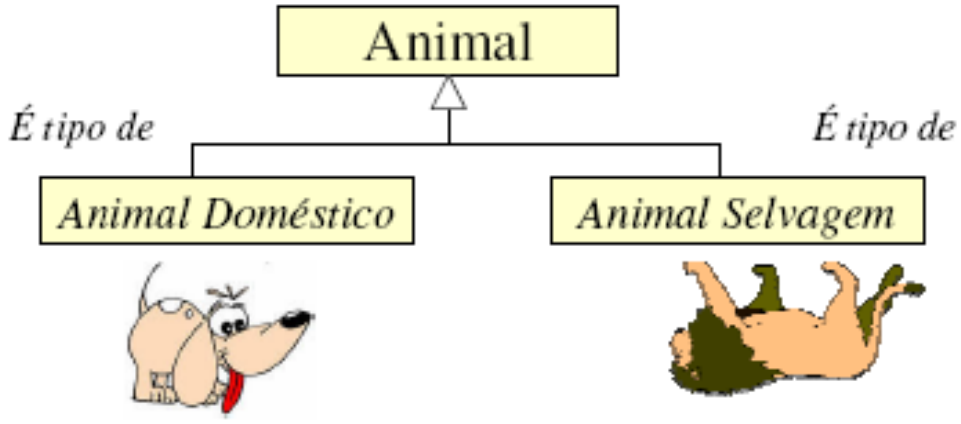
Herança é o mecanismo pelo qual elementos mais específicos incorporam a estrutura e comportamento de elementos mais gerais.

Uma classe derivada **herda** a estrutura de **atributos** e **métodos** de sua classe "base", mas pode seletivamente:

- *adicionar novos métodos*
- *estender a estrutura de dados*
- *redefinir a implementação de métodos já existentes*

Uma classe "pai" ou *super classe* proporciona a funcionalidade que é comum a todas as suas classes derivadas, filhas ou *sub classe*, enquanto que uma classe derivada proporciona a funcionalidade adicional que especializa seu comportamento.

Exemplo:



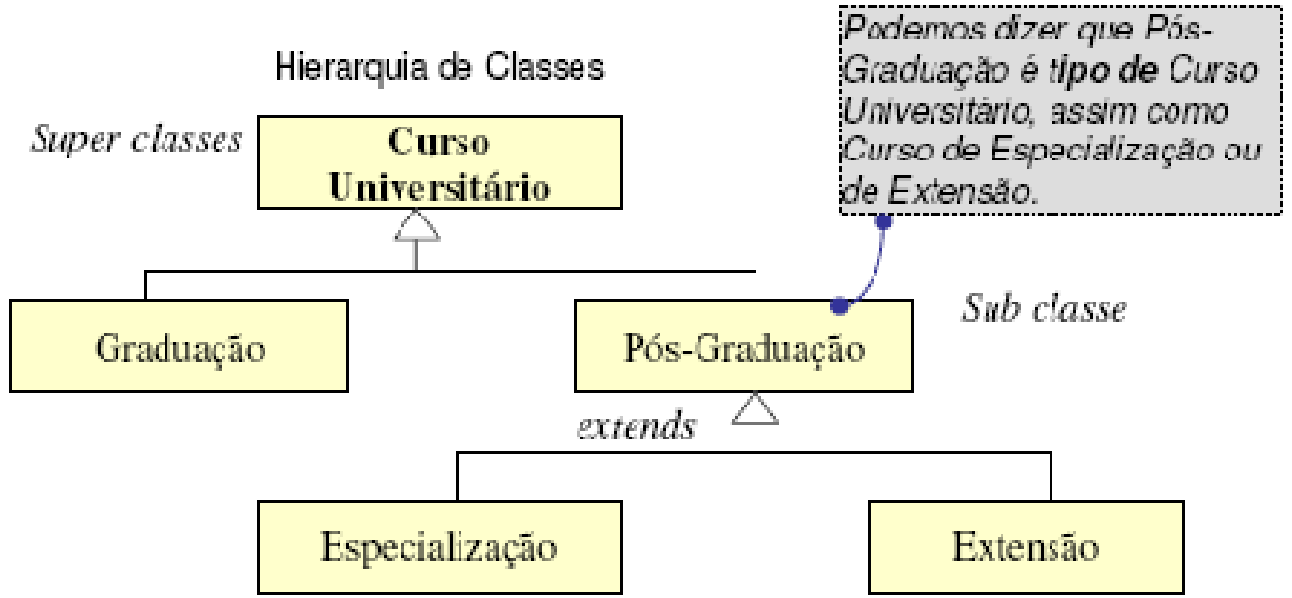
Orientação a Objetos

Herança

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Herança

Exemplo 1

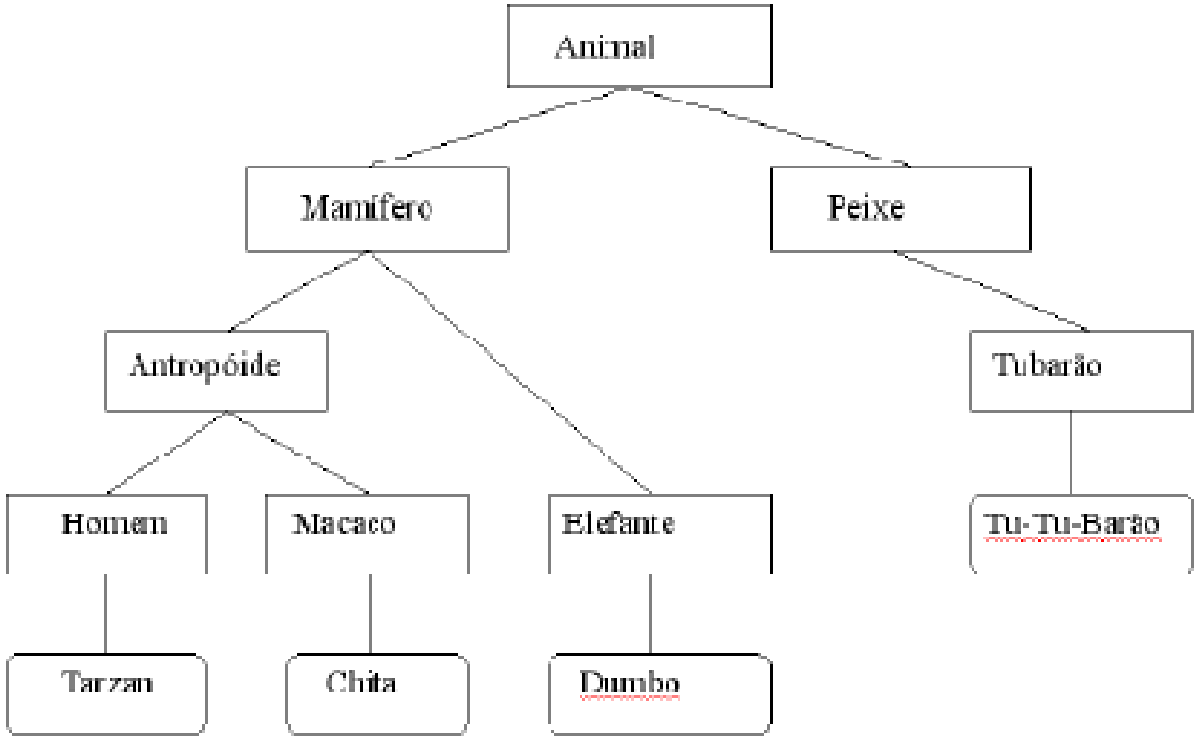


Herança

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Herança

Exemplo 2



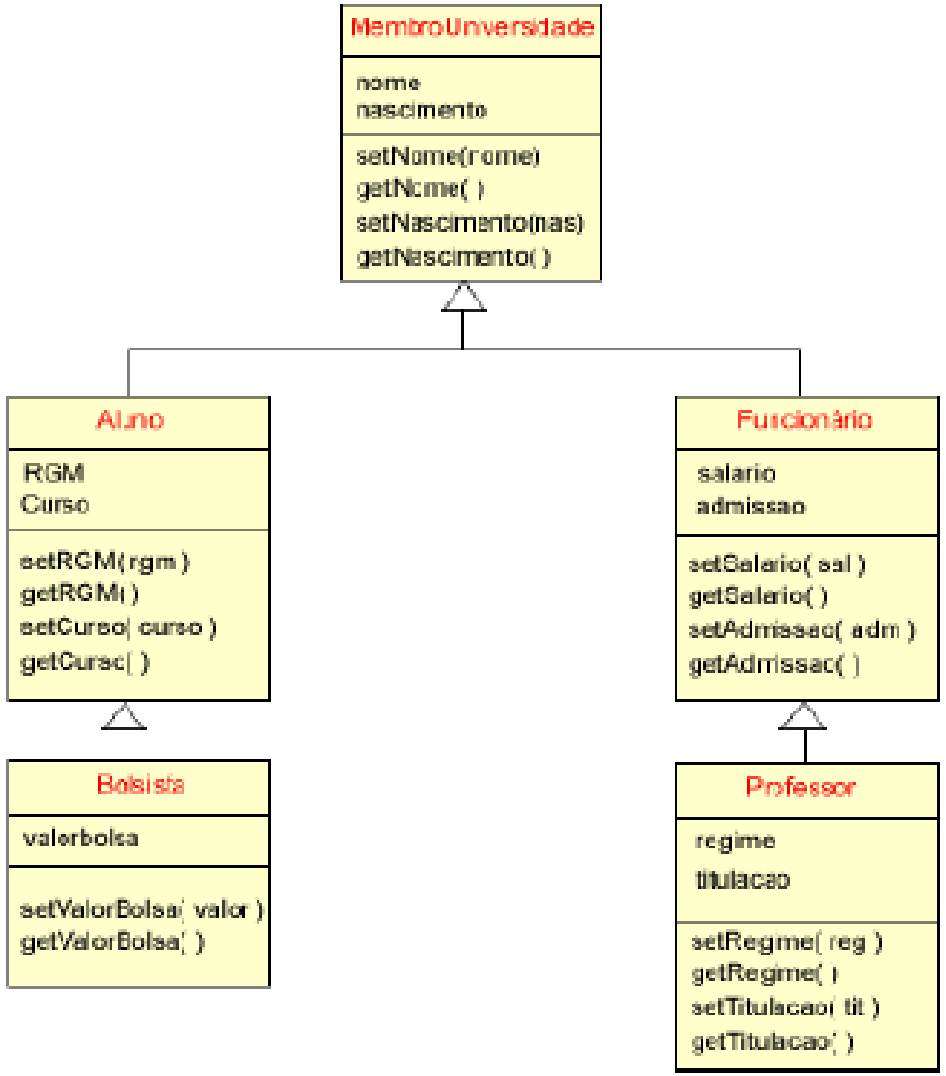
Cada subclasse é uma candidata a ser super classe de outras classes

Herança

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Herança

Exemplo 3



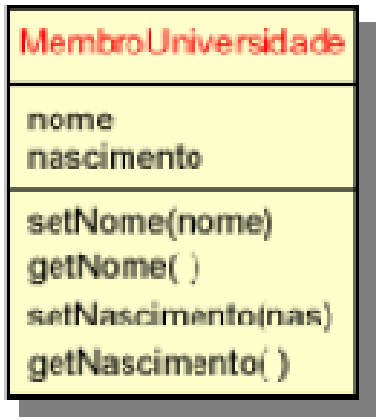
Orientação a Objetos

Herança

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

```
public class MembroUniversidade
{
        private String nome;
        private String nascimento;

        public MembroUniversidade(String nnome, String nnascimento) {
                nome = nnome;
                nascimento = nnascimento;
        }

        public MembroUniversidade() {
                nome = "";
                nascimento = "";
        }

        public void setNome(String nnome) {
                nome = nnome;
        }

        public String getNome() {
                return nome;
        }

        public void setNascimento(String nnascimento) {
                nascimento = nnascimento;
        }
}
```

Construtores



Orientação a Objetos

Herança

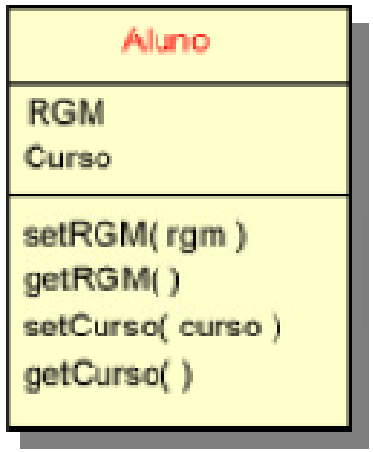
- *Classes*
- *Objetos*
- *Atributos*
- *Métodos*
- *Abstração de Dados*
- ***Herança***
- *Polimorfismo*
- *Encapsulamento*
- *Associação*
- *Interface*

Palavra-chave que indica herança

super classe

Chama o construtor da classe pai

```
public class Aluno extends MembroUniversidade
{
        private String RGM;
        private String curso;

        public Aluno(String no, String na, String r, String cur) {
                super(no, na);
                RGM = r;
                curso = cur;
        }

        public Aluno()  {
                super( );
                RGM = "";
                curso = "";
        }

        public void setRGM(String r) { ... }

        public String getRGM( ) { ... }

        public void setCurso(String cur) { ... }

        public String getCurso( ) { ... }
}
```



Orientação a Objetos

Herança

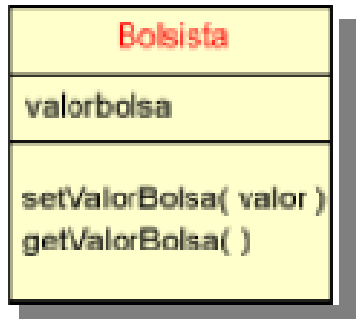
- *Classes*
- *Objetos*
- *Atributos*
- *Métodos*
- *Abstração de Dados*
- ***Herança***
- *Polimorfismo*
- *Encapsulamento*
- *Associação*
- *Interface*

Palavra-chave que indica herança

super classe

```
public class Bolsista extends Aluno
{
        private float valorbolsa;

        public Bolsista(String no, String na, String r, String cur, float v) {
                super(no, na, r, cur);
                valorbolsa = v;
        }

        public Bolsista() {
                super( );
                valorbolsa = 0;
        }

        public void setValorBolsa(float v) { ... }

        public float getValorBolsa( ) { ... }

}
```

Chama o construtor da classe pai



Orientação a Objetos

Herança

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Podemos instanciar objetos de qualquer uma das classes

```
MembroUniversidade a = new MembroUniversidade("Maria","15/05/1971");
MembroUniversidade b = new MembroUniversidade();
```

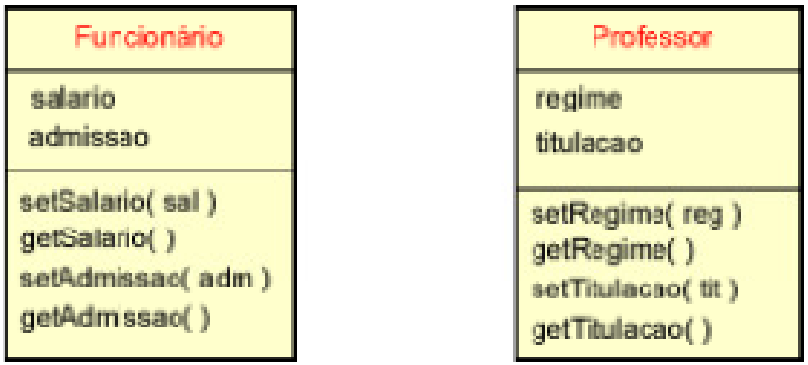
```
Aluno a = new Aluno("Maria","15/05/1971","34.555-6","TSI");
Aluno b = new Aluno();
```

```
Bolsista a = new Bolsista("Maria","15/05/1971","34.555-6","TSI",180.50f);
Bolsista b = new Bolsista();
```

Herança

Exercício

Implemente as classes Funcionario e Professor conforme o modelo anterior.



Orientação a Objetos

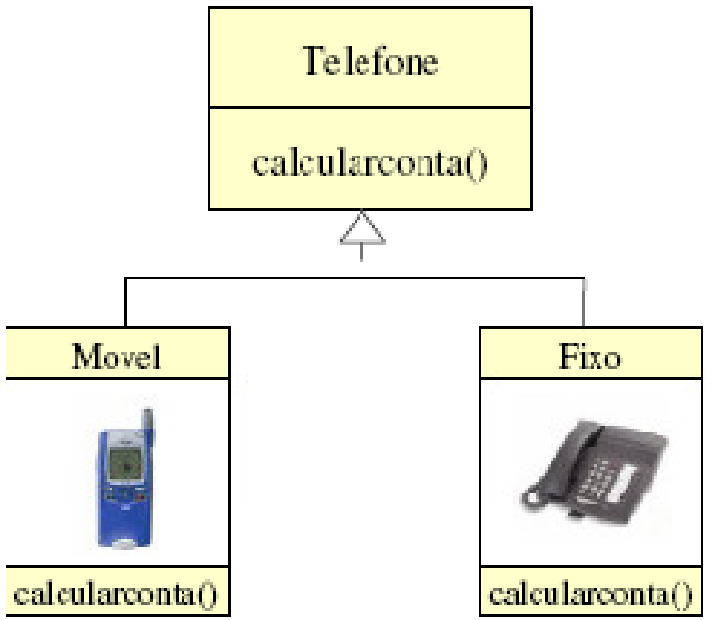
Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Polimorfismo:

Definição:
*"**Polimorfismo**" é uma operação que pode assumir múltiplas formas, a propriedade segundo o qual uma operação pode comportar-se diferentemente em classes diferentes"*
O polimorfismo é o responsável pela extensibilidade em programação orientada a objetos, promovendo o reuso.

Exemplo:



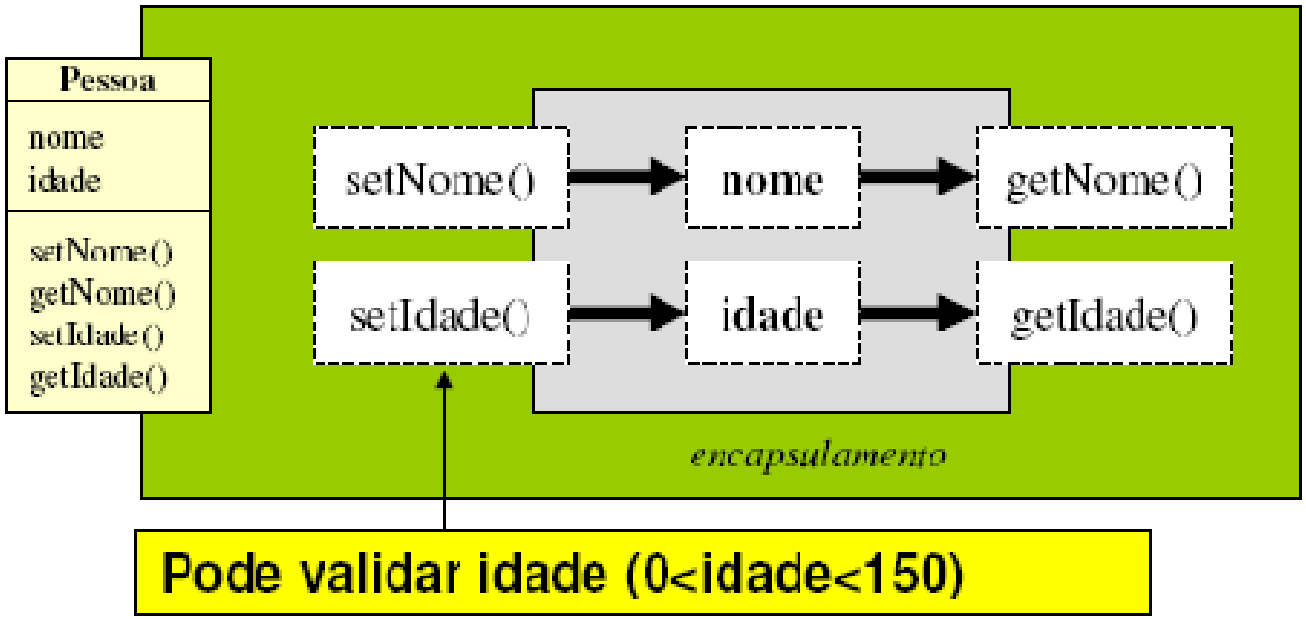
Orientação a Objetos

Principais Conceitos

Classes
Objetos
Atributos
Métodos
Abstração de Dados
Herança
Polimorfismo
Encapsulamento
Associação
Interface

Encapsulamento:

Benefícios
- **Segurança:**
  Protege os atributos dos objetos de terem seus valores corrompidos por outros objetos.
- **Independência:**
  "Escondendo" seus atributos, um objeto protege outros objetos de complicações de dependência de sua estrutura interna



Exemplo Encapsulamento

```
class Pessoa {
    private int idade;
    protected String cpf;
    public String nome;
    Pessoa() {
                this.idade = 0;
                this.nome = "";
                this.cpf = "";
    }
    public void setIdade(int idade) {
       this.idade = idade;
    }
    public int getIdade() {
       return (idade);
    }
    public void setNome(String nome) {
                this.nome = nome;
    }
    public String getNome() {
       return (nome);
    }
    public void setCpf(String cpf) {
                this.cpf = cpf;
    }
    public String getCpf() {
       return (cpf);
    }
}
```

public: pode ser acessado mesmo de outra classe
private: só pode ser acessado da própria classe
protected: acessado pela própria classe e por classes derivadas

Exemplo Encapsulamento

```
class Pessoa {
    private int idade;
    protected String cpf;
    public String nome;
    .....
}

public class Aluno extends Pessoa {

    public static void main( String args[]) {
        Pessoa aluno = new Pessoa();
        aluno.idade = 30;
        aluno.setIdade(25);
        System.out.println("Idade do Aluno "+aluno.getIdade() );
        aluno.nome = "José da Silva";
        aluno.setNome("José da Silva");
        System.out.println( "Nome do Aluno: "+aluno.nome );
        System.out.println( "Nome do Aluno: "+aluno.getNome() );
        aluno.cpf = "100.100.100-10";
        aluno.setCpf("100.100.100-10");
        System.out.println("CPF do Aluno: "+aluno.cpf);
        System.out.println("CPF do Aluno: "+aluno.getCpf() );
    }
}
```

public: pode ser acessado mesmo de outra classe
private: só pode ser acessado da própria classe
protected: acessado pela própria classe e por classes derivadas

Declaração da classe Aluno, que **herda** da classe Pessoa

Erro !!! Tentativa de acesso à um atributo **PRIVATE** da superclasse

Chamada ao método **SetIdade** da superclasse, que modifica a **idade**

Chamada do método **getIdade**, que retorna o valor atual do atributo **idade**

Acesso direto (desprotegido) ao atributo **nome** da superclasse

Acesso ao atributo **nome** através do método **getNome** (acesso protegido)

FIM

Referência: http://www.geocities.com/luccavallari/ievolution/