LPOO

Edeyson Andrade Gomes
edeyson@ifba.edu.br

Agenda

- Introdução
  - Paradigmas
  - Reuso de Software
- Definições Básicas
  - Abstração
  - Objeto
  - Classe

Paradigmas

Paradigma

- Paradigma (do grego parádeigma) literalmente modelo, é a representação de um padrão a ser seguido
  - http://pt.wikipedia.org/wiki/Paradigma

- Paradigmas de Programação
  - Imperativa
    - Fortran, Pascal, C
  - Orientada a Objetos
    - Java, C++, C#
  - Funcional
    - LISP
  - Lógica
    - Prolog

Paradigma

- O que é comum às Linguagens de Programação?
  - Devem possuir Sintaxe e Semântica bem definidas;
  - Devem ser executáveis com eficiência;
  - Devem possibilitar a expressão de qualquer problema computável.

- O que diferencia as Linguagens de Programação?
  - Propósito;
    - Fortran x Cobol.
  - Avanços Tecnológicos;
    - C#
  - Interesses comerciais;
    - .NET x Java
  - Cultura;
  - **Paradigma**.

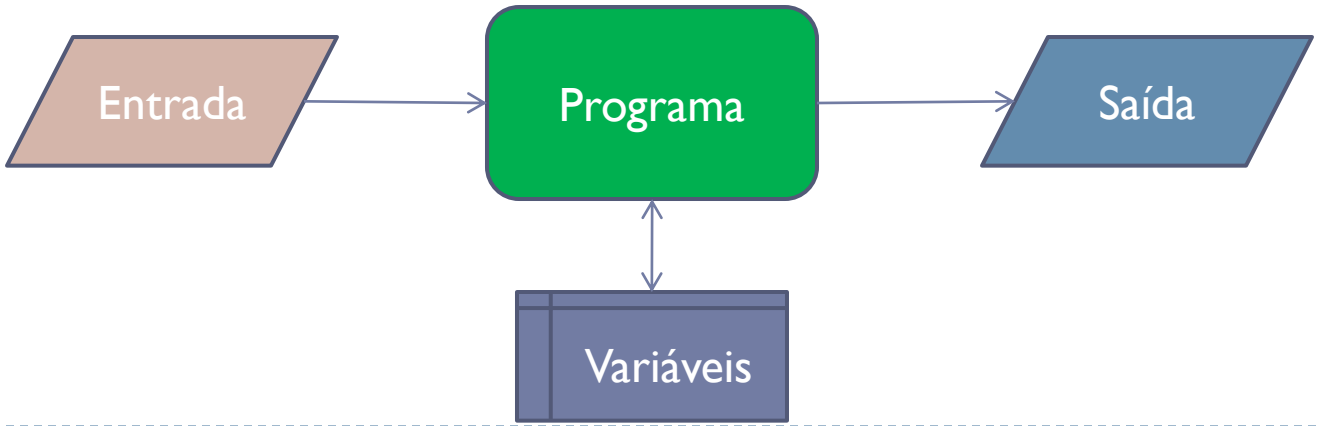
Paradigma de Programação

- Modelo ou padrão de programação.
- Arquitetura de Von Newmann
  - Direção para projeto de linguagens de programação
    - Variáveis, comandos de atribuição, execução sequencial de instruções e repetição iterativa.

Paradigma Imperativo

Paradigma Imperativo

- Paradigma Imperativo
  - Programas centrados no conceito de um estado (modelado por variáveis) e ações (comandos) que manipulam o estado
  - Mais antigo e ainda mais usado
  - Modelo Computacional:



Paradigma Imperativo

- Vantagens
  - Eficiência (embute modelo de Von Neumann)
  - Modelagem simples de aplicações do mundo real
  - Paradigma dominante e bem estabelecido
- Desvantagens
  - Relacionamento indireto entre E/S resulta em:
    - Difícil legibilidade e manutenibilidade
    - Difícil evolução do software com ciclo de vida curto
    - Acoplamento Alto x Coesão

Paradigma Funcional

Paradigma Funcional

- Programas são funções que descrevem uma relação explícita e precisa entre Entrada e Saída
- Estilo declarativo
  - Não há o conceito de estado nem comandos como atribuição
- Conceitos sofisticados como polimorfismo, funções de alta ordem e avaliação sob demanda
- Aplicação
  - Prototipação em geral e Inteligência Artificial

Paradigma Funcional

- Modelo Computacional:

- Exemplos em LISP:

```
(defun fatorial (n)
  (do ((i n (- i 1))
       (resultado 1 (* resultado i)))
      ((= i 0) resultado)))
```

Paradigma Funcional

- Definindo uma função:

```
> (defun foo (x y) (+ x y 5))
FOO
> (foo 5 0)                ;chamando a função
10

> (defun somatorio (fun a b)
  (if (> a b) 0 (+ (funcall fun a)
     (somatorio fun (1+ a) b))))

> (somatorio (function quadrado) 1 4)
30
```

Paradigma Lógico

Paradigma Lógico

- Programas são relações entre Entrada e Saída
- Estilo declarativo, como no paradigma funcional
- Na prática, inclui características imperativas, por questão de eficiência
- Aplicações: sistemas especialistas e banco de dados
- Modelo Computacional:

Paradigma Lógico

- Quando o interpretador recebe uma consulta, ele tenta encontrar predicados que se encaixam na consulta, sejam eles fatos diretos ou regras que possuem o termo consultado como conclusão. Por exemplo:
  - irmaos(X,Y) :- filho(X,Z), filho(Y,Z).
  - que em Lógica de Primeira ordem é:
    XYZ((filho(X,Z)^filho(Y,Z))irmaos(X,Y))

Paradigma Lógico

```
filho(X,Y) :- pai(Y,X).
filho(X,Y) :- mae(Y,X).
mae(marcia, ana).
pai(tomas, ana).
pai(tomas, erica).
pai(marcos, tomas).
```

- De acordo com essa base, a seguinte consulta é avaliada como verdadeira:

```
?- irmaos(ana, erica).
    yes.
```

Paradigma OO

Paradigma OO

- Difere do Imperativo principalmente na forma de concepção e modelagem do sistema
- Uma aplicação é estruturada em módulos (classes) que agrupam um estado e operações (métodos) sobre este
- Classes podem ser criadas pelo usuário, estendidas e/ou usadas como tipos

Paradigma OO

- Vantagens
  - Todas as do estilo imperativo
  - Classes estimulam o projeto centrado em dados
    - Aumenta a modularidade, reusabilidade e extensibilidade
  - Aceitação comercial crescente

- Problemas
  - Semelhantes às do paradigma imperativo, mas amenizadas pelas facilidades de estruturação

Reuso de Software

Reuso de Software

- Consumo de combustível, navegação, compensação de forças externas e controle de velocidade, tudo é comandado por programas num AirBus 330.
  - É o conceito de fly-by-wire, ou vôo por fios, que mantém as aeronaves no ar, monitora os sistemas e guia os "administradores" mesmo na escuridão das tempestades.
  - Quanto maior o número de linhas, mais complexo é o software e, portanto, mais sujeito a conflitos e falhas de execução.
    - Imagine isso com 10.000.000 de linhas de código.

Reuso de Software

- Problemas na Produção de Software
  - Complexidade
    - Tamanho de sistemas, complexidade de requisitos
  - Custo
    - Pessoal capacitado
  - Tempo
    - Software novo é produzido de forma lenta e com prováveis erros (bugs)

- ***Solução***:
  - REUSO
    - Experiência

Reuso de Software

- **Dificuldades da Solução**
  - Efetividade no Reuso
    - Como alcançar? Que técnicas?
  - O Que reusar X Como reusar
    - Entendimento dos técnicos e interessados
  - Custos e *Overhead*
    - Curva de aprendizado
  - Orientação a Objetos
    - Promessa x Resultados
    - Dificuldades

Tipos de Reuso

Tipos de Reuso

- 1960 – 1970
  - Linhas de código, Subrotinas e Bibliotecas
- 1980
  - Orientação a Objetos
    - Herança, composição, delegação, polimorfismo
  - Reuso de Abstrações
  - Composição x Herança
    - Melhor Reuso
  - TIPO (ADT)
    - Acoplamento Fraco
    - Programar para Interfaces, não para Implementações

Tipos de Reuso

- Técnicas que Dificultam o Reuso
  - Criar um objeto (uma variável, por exemplo) especificando seu tipo explicitamente
    - O código se compromete com uma implementação particular

- Polimorfismo
  - Melhor reuso
    - Tipo determinado em tempo de execução
    - *Late Binding*

Tipos de Reuso

- Componentização
  - Reutilização de objetos inteiros
  - Abstração
  - Objetos manipuláveis em tempo de projeto
    - Tempo de execução X projeto
  - Revolução do RAD (*Rapid Application Development*)

Tipos de Reuso

- **Técnicas para melhorar código para aumentar a qualidade do software**
  - Princípio Aberto-Fechado
    - Manutenção fácil e reuso
  - Incluindo sua legibilidade, robustez, reusabilidade
  - *Refactoring*
    - Fazer mudanças de *forma* a um programa *sem* introduzir mudanças funcionais

Princípio Aberto-Fechado

- Código bem projetado pode ser estendido sem a necessidade de modificação
  - Adição de funcionalidade via adição de código
- Mecanismos Primários
  - Abstração e Polimorfismo

Abstração

Abstração

- "É o ato pelo qual separamos, num objeto, uma qualidade particular para considerá-la isoladamente de todas as outras, e com exclusão do próprio sujeito.

  - Consiste em separar (*abstrahere* = arrancar, desligar) pelo pensamento, ou considerar separadamente, o que não pode ser dado separadamente, na realidade.

  - Abstrair é separar atributos, elementos. O raciocínio humano age por abstrações."

    - http://www.filoinfo.bem-vindo.net/filosofia/modules/lexico/entry.php?entryID=64

Abstração

- "Abstração não deve ser confundida com a análise.
- A análise considera igualmente todos os elementos da representação analisada, e divide em partes uma coisa composta;
  - considera, isoladamente, uma qualidade comum a uma multidão de compostos.
  - Assim reconhecer a brancura de uma rosa determinada é fazer análise;
  - conceber a brancura em si mesma, como qualidade peculiar a um grande número de objetos, é proceder abstração.
- A abstração é, portanto, a base da formação das ideias gerais"
  - http://www.filoinfo.bem-vindo.net/filosofia/modules/lexico/entry.php?entryID=64

Abstração

- "Abstração é o processo ou resultado de generalização por redução do conteúdo da informação de um conceito ou fenômeno observável, normalmente para reter apenas a informação que é relevante para um propósito particular.
  - Por exemplo, abstraindo uma bola de futebol de couro, por uma bola de futebol, retemos apenas a informação enxuta das propriedades e comportamentos da palavra.
    - Cientistas da computação usam a abstração para entender, resolver problemas e comunicar suas soluções para o computador usando alguma linguagem computacional em particular."

      - http://pt.wikipedia.org/wiki/Abstra%C3%A7%C3%A3o

Abstração

- O que é uma esfera?
  - Que atributos possui?
- O que é uma mesa?
- Como fazemos para descrever um objeto do mundo real?
- Todo objeto do mundo real é concreto?
- O que é uma Planta Baixa? Como um arquiteto apresenta uma Casa a ser construída?

Objeto

Objeto

- Sendo cada um de nós observadores do Mundo Real, o que vemos? O que tocamos? O que sabemos existir?
  - Árvore, carro, relógio, ar, poluição, som, etc.

- Tudo aquilo considerado exterior, física ou psicologicamente, ao observador é denominado de **objeto**.
  - Coisa seria um termo ruim ☺

Objeto

- Toda Mesa é um objeto.
  - Todo objeto tem propriedades, atributos.
  - Por exemplo: largura, altura, peso.

- Existem inúmeros tipos diferentes de mesa. Como identificamos um objeto como mesa?
  - Nós abstraímos as propriedades básicas de uma mesa.
  - Fazemos analogia, comparação.
  - Concluímos, pela análise de propriedades, que um objeto classifica-se como mesa.

Objeto

- Toda Cadeira é um objeto.
  - Como objeto, cadeira tem propriedades, atributos.
  - Por exemplo: largura, altura, peso, número de pés, material.

- Existem inúmeros tipos diferentes de cadeira. Como identificamos um objeto como cadeira?
  - O processo de abstração do cérebro é o mesmo.

Objeto

- Toda TV é um objeto.
  - Como objeto, TV tem propriedades, atributos.
  - Por exemplo: largura, altura, peso.
  - Mas, pode-se mudar o canal, mudar o volume, mudar o brilho, etc.

- O que TV tem que cadeira e mesa não possuem?
  - Resposta: Comportamento

Objeto

- Um objeto não possui apenas atributos. Ele pode possuir, também, comportamento.
  - TV tem comportamentos para manipular som, canal, brilho, etc.
  - Como descrever um carro?
    - Atributos: placa, Chassi, marca, modelo, cor, HP, etc.
    - Comportamento: acelerar, frear, mudar marcha, etc.
  - Como descrever um aparelho de ar-condicionado?

Objeto

- Atributos e Estado
  - Os valores dos Atributos de um Objeto determinam seu Estado
  - O Estado de um Objeto pode variar, logo, os valores dos Atributos podem mudar
  - Normalmente quem muda o Estado de um Objeto é um de seus Comportamentos

  - Por exemplo:
    - Carro tem atributo *Quilometragem* que muda quando este *Anda*. Veja que *Andar* é um comportamento do Carro.

Objeto

- Exercício
  - Determine Objetos com Atributos e Comportamentos.
  - Indique quando e como o Estado destes Objetos pode mudar.
  - Exemplifique.

Classe

Classes

- Vamos fazer um novo exercício
  - Observe o mundo real
  - Todos os objetos percebidos são classificados em Tipos diferentes
  - Por exemplo:
    - Costumamos Classificar Carro, Moto, Caminhão e Ônibus como Veículos
    - Poderíamos classificar os Veículos como Veículos Leves e Veículos Pesados (ainda, Utilitários ou de Passeio)

Classes

- Vamos fazer um novo exercício
  - Lembremos da biologia e vamos **classificar** um Cachorro
    - **Espécie:** *Canis familiaris*
    - **Gênero:** Canis
    - **Família:** Canidae
    - **Ordem:** Carnívora
    - **Classe:** Mammalia
    - ...
  - Vejam que o que diferencia um Lobo de um Cachorro é a Espécie**:** *Canis Lupus*

Classes

- **Taxonomia**
  - Classificação de coisas ou objetos animados, inanimados, lugares e eventos
    - Tudo pode ser classificado de acordo com algum esquema taxonômico.
  - Podemos usar duas visões (metodologias) para modelar (Classificar) o Mundo Real:
    - I. De Cima para Baixo
      - Do mais geral para o mais específico
    - II. De Baixo para Cima
      - Do mais específico para o mais geral

Classes

- 1. Classificação *de Cima para Baixo*
  - Pensar nos atributos e comportamentos de qualquer veículo
  - Por exemplo:
    - Atributos: Peso, Largura, Comprimento, Altura, Combustível, Velocidade Máxima, Consumo por Litro
    - Comportamento: Acelerar, Frear, Trocar Marcha
  - Determinar o que é específico para cada subtipo
    - Todo Carro é um veículo, logo, todo carro tem os atributos e comportamentos de veículo
    - Veículos Leves
      - Número máximo de Passageiros no banco traseiro
    - Veículos Pesados
      - Pinhão
      - Coroa
      - Longarina
      - Eixo Cardan

Classes

- Podemos pensar no Tipo geral e determinar o que é específico em seus subtipos

Classes

- 11. Classificação *de* Baixo para Cima
  - Pensar nos atributos e comportamentos de cada veículo particular
  - Depois, ver o que é comum e criar um Tipo superior

- Vamos discutir um pouco mais de classes revendo um programa Pascal:

Tipos em Pascal

```
PROGRAM TiposEmPascal;
USES CRT;

TYPE    Aluno = RECORD
     nome: string;
     idade: integer;
     notas: array [1..3] of real;
END;

VAR    aluno1 : Aluno;
       media, soma: real;
       i : integer;
```

- Ponto crucial
  - Definição de TIPO
  - TIPO definido pelo Usuário
  - ***VAR aluno 1 : Aluno;***
  - Podem-se declarar novas variáveis do TIPO que o usuário definiu.
    - O que podemos fazer com estas variáveis?
      - Quais suas operações?
        - Vejamos o código.

Tipos em Pascal

```
PROGRAM   TiposEmPascal;
USES CRT;

TYPE   Aluno = RECORD
     nome: string;
     idade: integer;
     notas: array [1..3] of real;
END;

VAR   aluno1 :  Aluno;
      media, soma: real;
      i : integer;
```

```
BEGIN
      aluno1.nome := 'Washington Luis';
      aluno1.idade := 21;
      aluno1.notas[1] := 10.0;
      aluno1.notas[2] := 8.5;
      aluno1.notas[3] := 9.5;

      FOR i := 1 TO 3 DO
          soma := soma + aluno1.notas[i];

      media := soma / 3;

      WRITELN('Aluno: ', aluno1.nome);
      WRITELN('Idade: ', aluno1.idade);
      WRITELN('Media:  ', media:3:2);

      READKEY;
END.
```

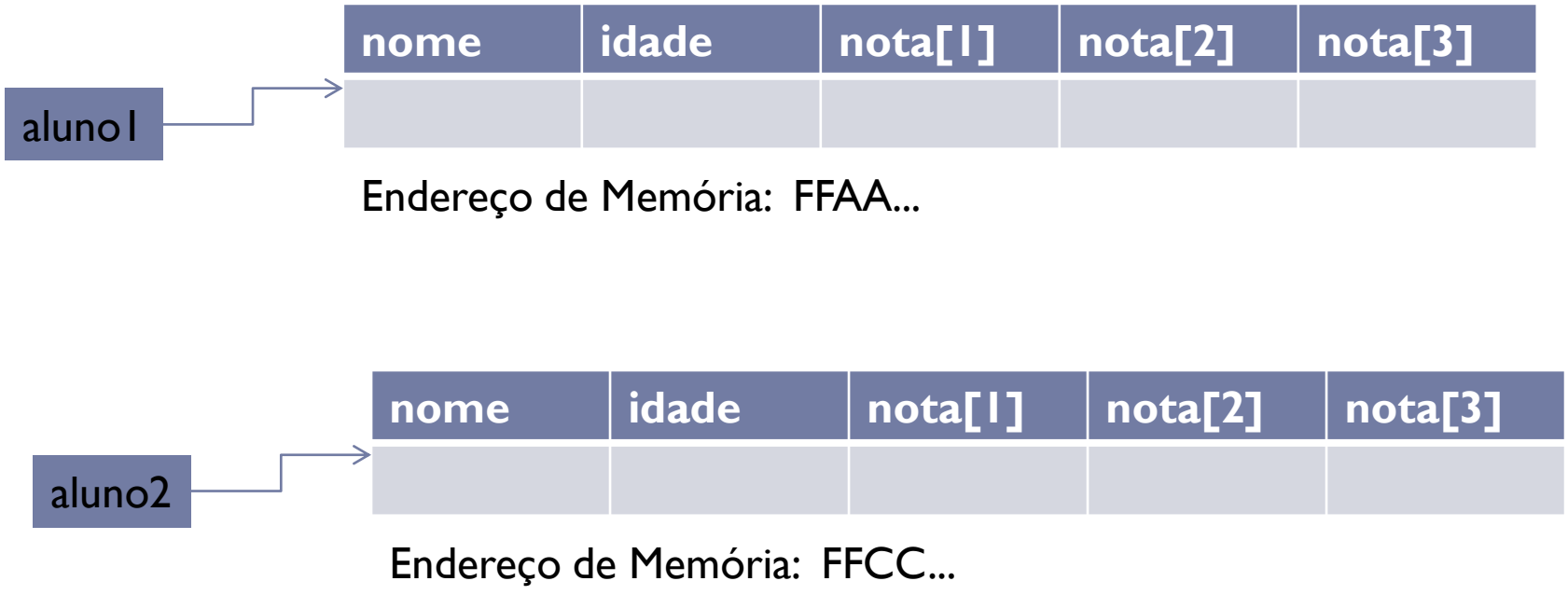
Tipos em Pascal

- Instanciação
  - Quando, em Pascal, definimos a variável aluno1 do TIPO aluno, ele aloca uma área de memória com o MODELO do TIPO Aluno
    - Assim, cria-se uma área de memória que caibam os atributos nome, idade e notas e seu endereço é associado à variável aluno1
    - VAR aluno1 : Aluno

Endereço de Memória: FFAA

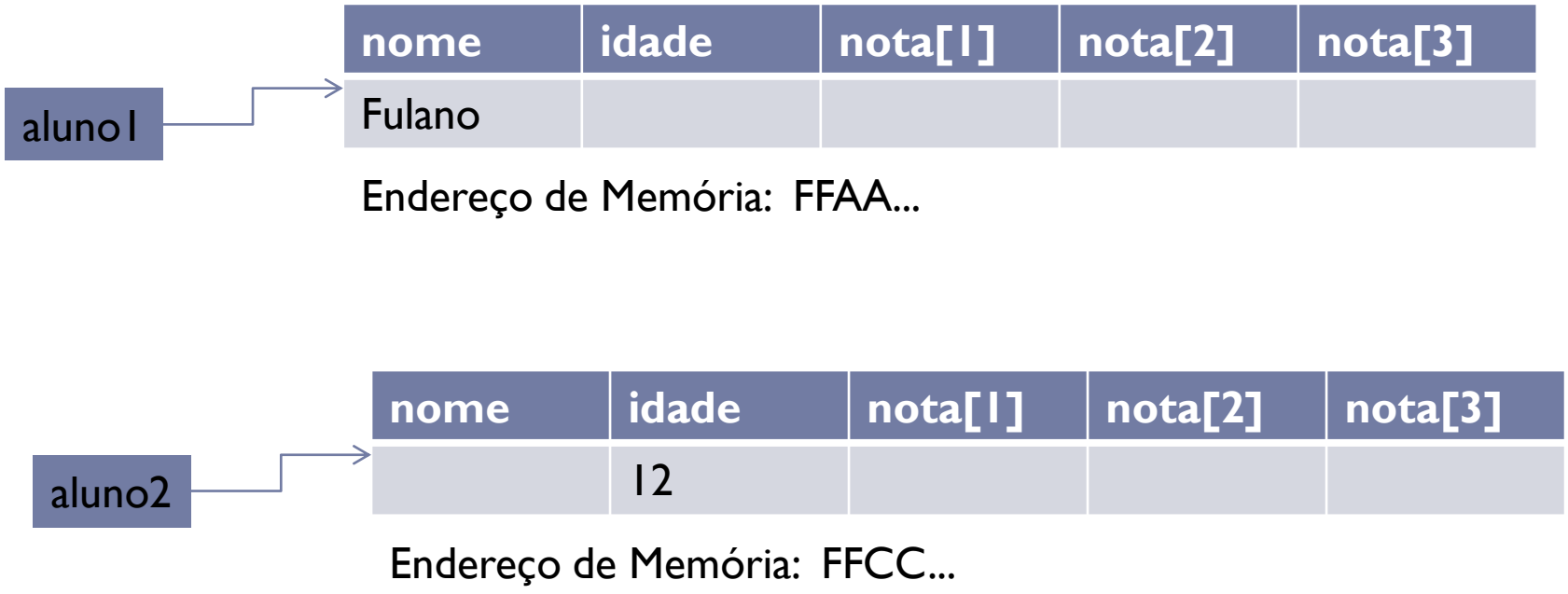
Tipos em Pascal

- Instanciação
  - Vejamos 2 variáveis do mesmo tipo:
    - VAR aluno1, aluno2 : Aluno



Tipos em Pascal

- Acesso
  - aluno1.nome = “Fulano”
  - aluno2.idade = 12



Tipos em Pascal

- Para cada TIPO definido pelo usuário em Pascal determinam-se apenas os atributos.
  - Ou seja, a **estrutura** do TIPO.
- Um TIPO definido pelo usuário em Pascal suporta as operações de atribuição e acesso aos atributos.

- Não existem níveis de acesso aos atributos. Todos são públicos.
  - Toda variável do TIPO tem todos os seus atributos visíveis ao programa.

Tipos em Java

- Vamos ver o Tipo Aluno definido em Java

```
class Aluno {
     String nome;
     int idade;
     double[ ] notas = new double [3];
}
```

Note que, independente da linguagem, definimos APENAS a estrutura do TIPO Aluno. Ou seja, seus Atributos.

Compare com o Pascal

```
TYPE Aluno = RECORD
     nome: string;
     idade: integer;
     notas: array [1..3] of real;
END;
```

Tipos em

```
public class Teste {


    public static void main(String[] args) {
        Aluno aluno1 = new Aluno();
        int i;
        double soma = 0, media;

        aluno1.nome = "Washington Luis";
        aluno1.idade = 21;
        aluno1.notas[0] = 10.0;
        aluno1.notas[1] = 8.5;
        aluno1.notas[2] = 9.5;

        for (i = 0; i < 3; i++) {
                    soma = soma + aluno1.notas[i];
        }

        media = soma / 3;

        System.out.println("Aluno: " + aluno1.nome);
        System.out.println("Idade: " + aluno1.idade);
        System.out.println("Media:  " + media);
    }
}
```

Estrutura

- Até aqui definimos a Estrutura para um novo Tipo
  - A estrutura consiste nos atributos do Tipo
  - Pascal e Java estão parecidos nisso?

- Vamos entender melhor Estrutura.
  - Quando definimos: `VAR aluno1 : Aluno;` o que dizemos ao Compilador é: defina a variável `aluno1` de tal forma que ela mantenha em memória um nome, uma idade e um vetor de notas.

- `Porém...`

Estruturas

- Suponha que queremos definir o TIPO Aluno com nome, idade e notas como feito.
  - Porém, nome só deve ter letras (nenhum caractere não alfabético é permitido) e a primeira letra de cada nome ou sobrenome deve ser maiúscula.
  - Como fazer isso em Pascal?
    - Criar uma função ou procedimento é solução?
    - Funções ou procedimentos podem impedir o programador de acessar o atributo nome de uma variável do TIPO Aluno?

Estruturas

- Uma solução ao Pascal seria a seguinte Função:

```
procedure setNome(a : Aluno; nome : string);
begin
      a.nome := nome;
      {Valida e formata o nome...}
end;
```

- Uma vez que o programador chame o procedimento setNome, este se encarrega de formatar corretamente o nome e o atribuir ao aluno.
  - Porém, nada impede o programador de não usar tal procedimento e atribuir um valor inválido ao nome.

Encapsulamento

- A Orientação a Objetos traz uma solução ao problema apresentado chamada de ENCAPSULAMENTO
- O encapsulamento para a OO significa esconder informação.
  - Objetiva-se proteger o estado interno do objeto (valores dos atributos), disponibilizando somente uma camada de acesso ao mesmo
    - Esta camada é implementada por métodos de acesso
      - GETs e SETs

Encapsulamento

- Para garantir o encapsulamento, a orientação a objetos (OO) define níveis de visibilidade para os atributos de um TIPO
  - PRIVATE – atributos acessíveis apenas dentro da classe
  - PUBLIC - atributos acessíveis dentro e fora da classe
  - PROTECTEC – atributos acessíveis dentro da classe, por suas subclasses e por todas as classes do mesmo pacote
  - Exemplo:

```
public class Pessoa {
    private int idade;
    protected String nome;
    public Endereço endereço;
}
```

Encapsulamento

```
public class Pessoa {
    private int idade;
    protected String nome;
    public Endereço endereço;
}
```

- A definição da classe pessoa **permite** as seguintes referências em outra classe:
  - Pessoa pessoa;
  - pessoa.nome = "José";
  - pessoa.endereço = new Endereço();

- Porém, a definição da classe pessoa **não permite** a seguinte referência em outra classe:
  - pessoa.idade = 12;

Encapsulamento

- Um exemplo radical:
  - Se eu quiser saber o que Selma almoçou, eu tenho duas formas de fazê-lo:
    - 1. Perguntando a ela
      - Respeito o encapsulamento
    - 2. Usando uma Peixeira
      - Destruo o encapsulamento
  - O que é melhor?
    - Assim também é com código. Respeitamos o encapsulamento da Classe e ela esconde seus detalhes.

Encapsulamento

- Mais uma razão
  - Se soubermos que o atributo nome é do tipo String, haverá um compromisso externo com o tipo do atributo e a Classe jamais poderá alterá-lo
    - Isso cria um forte acoplamento entre a classe quem a usa
  - Se criarmos um método de acesso, por exemplo, getNome, que retorna uma String, o nome precisa ser String? Ele pode mudar?
    - Se sim, teremos baixo acoplamento entre a classe e quem a usa

Comportamento da Classe

- Toda Classe define uma Estrutura e um conjunto de Comportamentos
  - O conjunto de comportamentos compõe sua INTERFACE
  - Comportamento é qualquer ação definida numa Classe através de um Método (função)
  - Métodos, assim como atributos, tem visibilidade PRIVATE, PROTECTED e PUBLIC

Instância de uma Classe

- Uma classe define um modelo de objeto
- Toda Classe compartilha uma mesma classe mãe: *Object*
- Para que um objeto passe a existir na memória segundo um modelo (Classe) é necessário criar uma instância (ocorrência de um objeto da Classe)
- Cada instância é identificada unicamente por um OID (Object ID)
  - Ao imprimir uma instância cuja Classe não define o método toString, obtem-se o OID da mesma
  - O método toString é usado para imprimir instâncias de uma classe e pode ser definido e especializado pelo usuário

Instância de uma Classe

- Suponha a classe TV:

```
public class TV {
    public int canal;
    protected int volume;
}
```

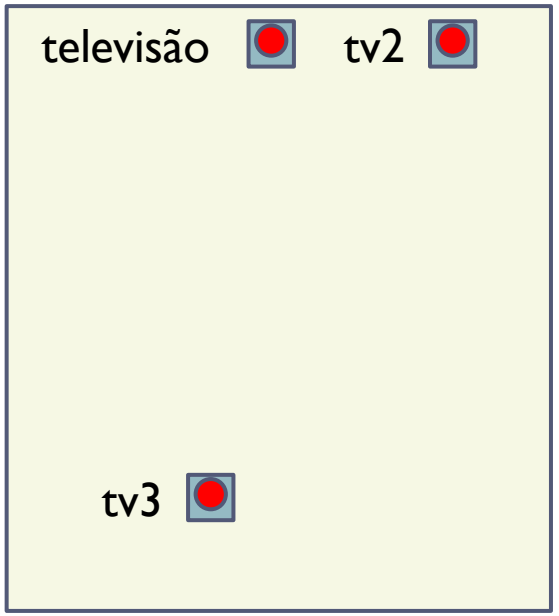
- Instanciar uma TV significa alocar área de memória com o modelo da Classe TV.
  - Sabendo que o tipo **int** ocupa 4 bytes, em Java teríamos o seguinte alocado na memória:

#ref	Canal (4 bytes)	Volume (4 bytes)

Instância de uma Classe

- Vejamos a classe de teste:

```
1  public class teste {
2
3      public static void main(String[] args) {
4          TV televisão, tv2, tv3;
5
6          televisão = new TV();
7          tv2 = televisão;
8
9          televisão.volume = 5;
10         tv2.canal = 11;
11         tv3 = tv2;
12
13         System.out.println(televisão.canal);
14         System.out.println(tv2.volume);
15     }
16 }
```

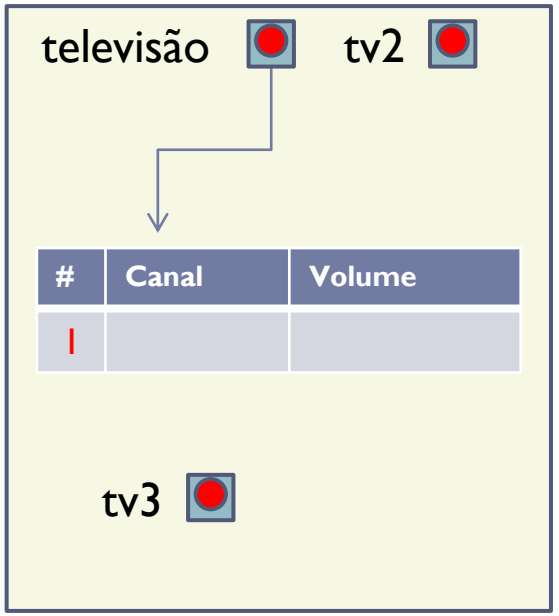


- A linha 3 cria um método de execução da classe.
- A linha 4 define 3 variáveis do tipo TV.
  - televisão, tv2 e tv3 são "ponteiros de memória" para instâncias da Classe TV.

Instância de uma Classe

- Vejamos a classe de teste:

```
1  public class teste {
2
3      public static void main(String[] args) {
4          TV televisão, tv2, tv3;
5
6          televisão = new TV();
7          tv2 = televisão;
8
9          televisão.volume = 5;
10         tv2.canal = 11;
11         tv3 = tv2;
12
13         System.out.println(televisão.canal);
14         System.out.println(tv2.volume);
15     }
16 }
```

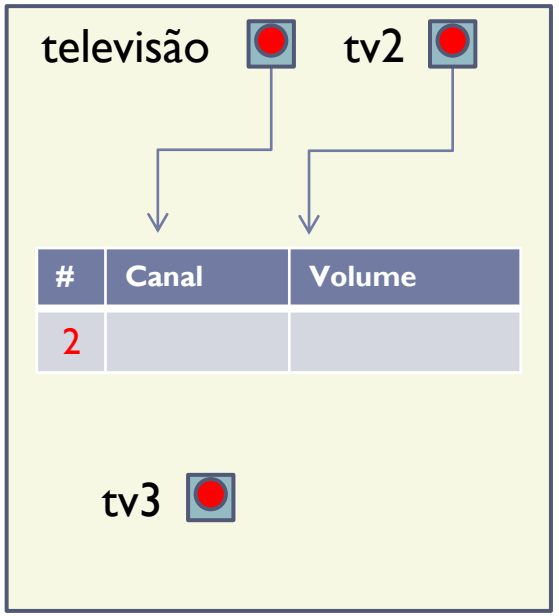


- A linha 6 cria uma instância da Classe TV e faz com que a variável televisão aponte para ela
  - Note que #ref muda para 1 informando quantas referências existem para a instância

Instância de uma Classe

- Vejamos a classe de teste:

```
1  public class teste {
2
3      public static void main(String[] args) {
4          TV televisão, tv2, tv3;
5
6          televisão = new TV();
7          tv2 = televisão;
8
9          televisão.volume = 5;
10         tv2.canal = 11;
11         tv3 = tv2;
12
13         System.out.println(televisão.canal);
14         System.out.println(tv2.volume);
15     }
16 }
```

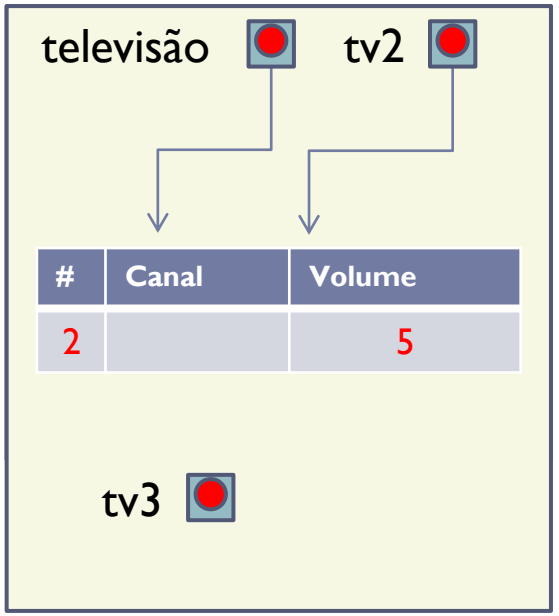


- A linha 7 cria faz com que a variável tv2 aponte para o mesmo endereço de memória apontado por televisão
  - Note que #ref muda para 2

Instância de uma Classe

- Vejamos a classe de teste:

```
1  public class teste {
2
3      public static void main(String[] args) {
4          TV televisão, tv2, tv3;
5
6          televisão = new TV();
7          tv2 = televisão;
8
9          televisão.volume = 5;
10         tv2.canal = 11;
11         tv3 = tv2;
12
13         System.out.println(televisão.canal);
14         System.out.println(tv2.volume);
15     }
16 }
```

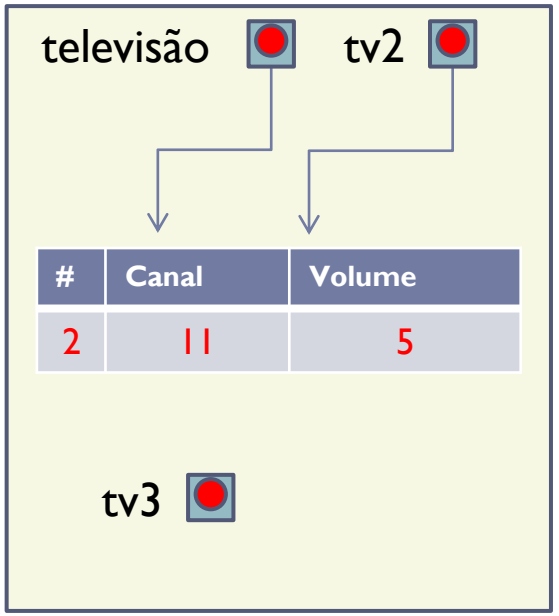


- A linha 9 faz com que o atributo volume (que é protegido) da instância apontada por televisão (e por tv2) mude para o valor 5

Instância de uma Classe

- Vejamos a classe de teste:

```
1  public class teste {
2
3      public static void main(String[] args) {
4          TV televisão, tv2, tv3;
5
6          televisão = new TV();
7          tv2 = televisão;
8
9          televisão.volume = 5;
10         tv2.canal = 11;
11         tv3 = tv2;
12
13         System.out.println(televisão.canal);
14         System.out.println(tv2.volume);
15     }
16 }
```

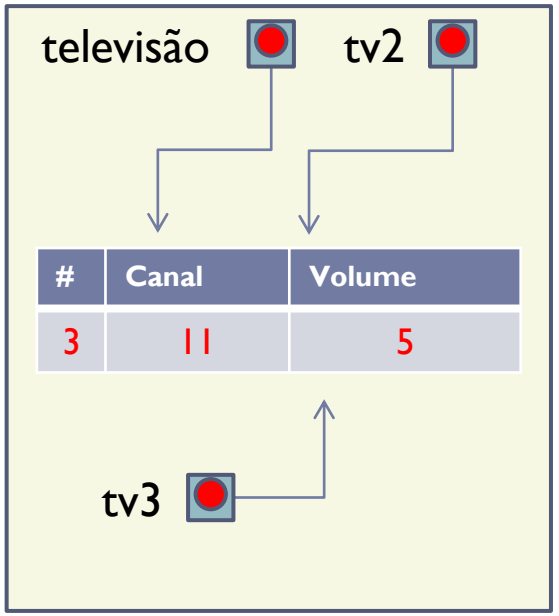


- A linha 10 faz com que o atributo canal (que é público) da instância apontada por tv2 (e por televisão) mude para o valor 11

Instância de uma Classe

- Vejamos a classe de teste:

```
1  public class teste {
2
3      public static void main(String[] args) {
4          TV televisão, tv2, tv3;
5
6          televisão = new TV();
7          tv2 = televisão;
8
9          televisão.volume = 5;
10         tv2.canal = 11;
11         tv3 = tv2;
12
13         System.out.println(televisão.canal);
14         System.out.println(tv2.volume);
15     }
16 }
```

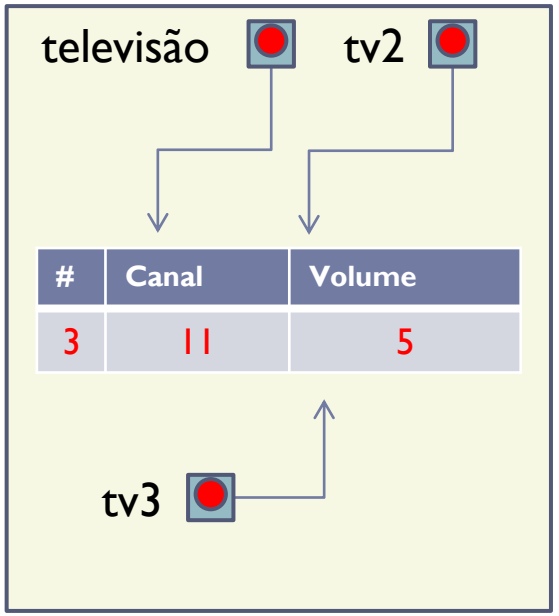


- A linha 11 faz com que a variável tv3 aponte para o mesmo endereço de memória apontado por tv2
  - Note que #ref muda para 3

Instância de uma Classe

- Vejamos a classe de teste:

```
1  public class teste {
2
3      public static void main(String[] args) {
4          TV televisão, tv2, tv3;
5
6          televisão = new TV();
7          tv2 = televisão;
8
9          televisão.volume = 5;
10         tv2.canal = 11;
11         tv3 = tv2;
12
13         System.out.println(televisão.canal);
14         System.out.println(tv2.volume);
15     }
16 }
```

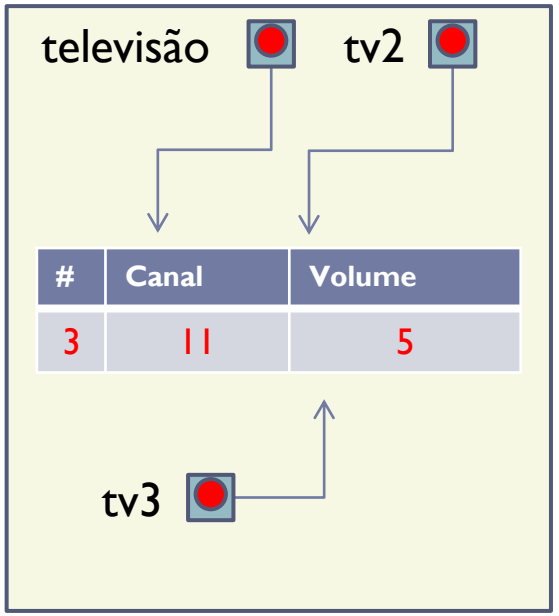


- A linha 13 imprime o valor do canal da instância referenciada pela variável televisão
- A linha 14 imprime o valor do volume da instância referenciada pela variável tv2

Instância de uma Classe

- Vejamos a classe de teste:

```
1  public class teste {
2
3      public static void main(String[] args) {
4          TV televisão, tv2, tv3;
5
6          televisão = new TV();
7          tv2 = televisão;
8
9          televisão.volume = 5;
10         tv2.canal = 11;
11         tv3 = tv2;
12
13         System.out.println(televisão.canal);
14         System.out.println(tv2.volume);
15     }
16 }
```



- Execução do programa:
  - 11
  - 5

Instância de uma Classe

- A *instanciação* de uma Classe passa por três etapas distintas:
  - 1. Declaração de variável com o TIPO complexo definido por uma Classe;
    - Sintaxe: *TIPO nomeDaVariável;*
    - Exemplo: *TV televisão;*
  - 2. Instanciação através da chamada ao operador *new*.
    - Exemplo: televisão = *new* …
  - 3. Inicialização através da chamada ao construtor da Classe.
    - Exemplo: televisão = *new* TV();

Instância de uma Classe

- Declaração de variável com o TIPO complexo definido por uma Classe;
  - A declaração de uma variável informa ao compilador que ela armazena valores do TIPO ou apontará para instâncias (áreas de memória) daquele TIPO.
    - So o tipo é complexo, a declaração faz da variável um ponteiro (*object reference*) para uma área de memória que armazenará o TIPO.
    - Se o tipo é primitivo (int, real, boolean, etc) a declaração já aloca a quantidade de memória necessária para a variável.

Instância de uma Classe

- Declaração de variável com o TIPO primitivo ou complexo definido por uma Classe:
  - Exemplos:
    - int *número*;
      - Aloca uma área de memória com 4 bytes e associa seu endereço ao símbolo *número*.
        - O compilador monta uma tabela com símbolos x endereços.
      - Fazer *número* = 14 significa colocar o valor 14 no endereço de memória guardado no símbolo *número* (via pesquisa na tabela).
    - TV tv1;
      - Aloca uma área de memória com 4 bytes (tamanho de um ponteiro - *object reference* - de memória com 32 bits) e associa seu endereço ao simbolo *tv1*.
      - Note que o conteúdo que *tv1* vai guardar é o endereço de memória onde estará uma instância da classe TV.
        - Nesta declaração, só de definição, *tv1* guarda um endereço nulo informando que ainda não aponta para uma instância.

Instância de uma Classe

- Instanciação através da chamada ao operador *new*.
- Inicialização através da chamada ao construtor da Classe.
  - O operador *new* instancia uma classe alocando um novo objeto na memória e retornando uma referência a seu endereço.
    - A chamada a *new* chama automaticamente o construtor da Classe.
    - *Instanciar uma Classe* é o mesmo que *criar um objeto*.
    - A chamada a new requer um argumento: uma chamada a um construtor, que é o mesmo nome da classe a instanciar

Objeto

- Todo Atributo ou Variável deve possuir um Tipo
- Além dos tipos primitivos (inteiro, real, boolean) a OO traz suporte a:
  - Tipos Complexos
    - Coleções (Conjuntos, Listas, Filas, Pilhas, etc)
    - String
    - Integer
  - Tipos Abstratos de Dados

Construtores

- Um construtor é um método usado pelo Java para instanciar um objeto
- Se uma classe não possui um construtor explícito, o Java define um implícito.
  - Por exemplo, a classe TV

```
public class TV {
    public int canal;
    protected int volume;
}
```

  - Ao fazermos tv2 = new TV(); chamamos o construtor implícito TV()

Construtores

- Um construtor é um método usado pelo Java para instanciar um objeto
  - Construtores, assim como métodos e atributos, têm visibilidade PRIVATE, PROTECTED ou PUBLIC
  - Pode ter ou não argumentos
    - É um método sobrecarregável
      - Mantém mesmo nome com assinaturas diferentes, ou seja, variando nome, quantidade e tipo de argumentos
  - *new* requer obrigatoriamente a especificação de um construtor.

Construtores

- Vejamos a classe TV:

```
public class TV {
    public int canal;
    protected int volume;

    /** Construtor sem argumentos    */
    public TV() {
        canal = 11;
        volume = 3;
    }

    /** Construtor sobrecarregado    */
    public TV(int canal, int volume) {
}
```

  - O construtor sem argumentos define, neste caso, valores padrão para os atributos da classe
    - Uma nova instância de TV criada com *new TV()* possui os atributos *canal* e *volume* com os valores 11 e 3, respectivamente.

Construtores

- Vejamos a classe TV:

```
public class TV {
    public int canal;
    protected int volume;

    /** Construtor sem argumentos    */
    public TV() {

    /** Construtor sobrecarregado    */
    public TV(int valorCanal, int valorVolume) {
        canal = valorCanal;
        volume = valorVolume;
    }
}
```

  - O construtor com argumentos recebe, neste caso, parâmetros (valores) para atribuir aos atributos da classe
    - Uma nova instância de TV criada com *new TV()* possui os atributos *canal* com valor valorCanal e *volume* com o valore valorVolume

Construtores

- Vejamos a classe TV:

```
public class TV {
    public int canal;
    protected int volume;

    /** Construtor sem argumentos    */
    public TV() {

    /** Construtor sobrecarregado    */
    public TV(int canal, int volume) {
        this.canal = canal;
        this.volume = volume;
    }
}
```

  - O construtor com argumentos recebe, neste caso, parâmetros (valores) para atribuir aos atributos da classe
    - Uma nova instância de TV criada com *new TV()* possui os atributos *canal* com valor *canal* e *volume* com o valore *volume*
      - O uso de *this* serve para diferenciar o atributo da classe (this.canal) do parâmetro do construtor (canal)

Construtores

- Curiosidade
  - Construtores não são Métodos – não possuem retorno
  - Veja que podemos ter métodos com mesmo nome e assinatura que o Construtor. Por exemplo:

1. TV tv1 = new TV();
2. tv1.TV();

```
public class TV {
    public int canal;
    protected int volume;

    /** Construtor sem argumentos    */
    public TV() {
        canal = 11;
        volume = 3;
    }

    /** Construtor sobrecarregado    */
    public TV(int canal, int volume) {
        this.canal = canal;
        this.volume = volume;
    }

    public void TV() {
        System.out.println("Método da TV");
    }
}
```

A linha 1 define a variável tv1 do tipo TV e a instancia via construtor TV().
A linha 2 faz a chamada ao método TV(), imprimindo: Método da TV