POO I

Vinícius Machado

Vetores e Matrizes

Vetores

- Estruturas de dados homogêneas que permite guardar vários valores (dados) em um mesmo identificador.
- Utilizamos a posição para identificar qual dos dados queremos acessar.

vetor

vetor[0]	vetor[1]	vetor[2]	vetor[3]

Vetores

- Para declarar um vetor indicamos seu tipo e o tamanho.

```
int[ ] vetor = new int[4];
int vetor[ ] = new int[4];
int vetor[ ] = {1, 2, 3, 4};
```

vetor[0]	vetor[1]	vetor[2]	vetor[3]
0	0	0	0
1	2	3	4

Matrizes

- Matrizes são, assim como vetores, estruturas de dados homogêneas que armazenam valores num formato n dimensional.
- Normalmente utilizamos o formato bidimensional (linha, coluna). No entanto, algumas vezes precisamos guardar mais valores, podendo declarar matrizes de 2, 3, 4, …, n dimensões.

Matrizes

- O formato comumente adotado para declararmos uma matriz é da seguinte maneira:

```
int mat[ ][ ] = new int[4][5];

int[ ][ ] mat = new int[4][5];
```

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]	mat[0][4]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]	mat[1][4]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]	mat[2][4]
mat[3][0]	mat[3][1]	mat[3][2]	mat[3][3]	mat[3][4]

Matrizes

- Ainda assim, é possível definirmos a quantidade de colunas após uma inicialização inicial das linhas. Por exemplo:

```
int values[ ][ ] = new int[5];
for (int i = 0; i < 5; i++) {
        values[ i ] = new int[ i * 3];
}
```

- Dessa forma podemos construirmos matrizes que não sejam necessariamente retangulares.

Vetores e Matrizes

- Tanto nas estruturas unidimensionais como n-dimensionais existe um atributo chamado length que permite acessar o tamanho dessa estrutura.
- Obs: tamanho != dados
- Dessa forma, podemos criar códigos que trabalhem com “qualquer” quantidade de dados sem precisarmos definir variáveis globais
    - Obs: variáveis globais são péssimas mas, raramente necessárias. Evitem!

Vetores - Exercício em aula

- Dado um vetor de números inteiros de tamanho definido pelo usuário, calcular:
  - Média
  - Mediana
  - Moda
  - Menor valor
  - Maior valor

- Que tal ordenarmos antes?

Matrizes - Exercícios em aula

- Crie um método capaz de printar uma matriz de qualquer tamanho.
- Faça um programa que chegue a estes padrões de desenho utilizando matrizes e utilize seu método para printá-las.

```
X O O O
X X O O
X X X O
X X X X
```

```
X X X X
X X X O
X X O O
X O O O
```

```
X X X X X
X O O O X
X O O O X
X X X X X
```

Classe Arrays

Como já falamos em aulas, JAVA possui muitas bibliotecas em sua API (veja a documentação). Estas bibliotecas aumentam nossa produtividade e diminuem a chance de erros em nossos programas.

A Classe Arrays possui vários métodos úteis para trabalharmos com vetores.

Arrays.sort(vetor[ ]);	Ordena um vetor em ordem crescente.
Arrays.fill (vetor[ ], value);	Preenche um vetor com o valor do parâmetro value.
Arrays.equal(vetor1[ ], vetor2[ ]);	Compara dois vetores retornando true/false
Arrays.binarySearch(vetor[ ], value);	Procura o valor value no vetor retornando sua posição. O vetor precisa estar ordenado. Caso o value não seja encontrado o resultado é menor que 0.

ArrayList

ArrayList

A API do Java fornece várias estruturas de dados pré definidas, chamadas coleções, utilizadas para armazenar grupos de objetos relacionados. Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados sem que seja necessário conhecer como os dados são armazenados.

A classe de coleção ArrayList<T> do pacote java.util fornece uma solução conveniente em tempo de execução para acomodar elementos adicionais - Ela altera dinamicamente seu tamanho.

<T> é o tipo do ArrayList

ArrayList<Integer>
ArrayList<String>

ArrayList

- Para declarar um ArrayList

```
ArrayList<String> nomesTurma;
```

- Para inicializar

```
nomesTurma = new ArrayList<String>();
```

- Pode ser na mesma linha

```
ArrayList<String> nomesTurma = new ArrayList<String>();
```

ArrayList

Método	Descrição
`add`	Adiciona um elemento ao fim de `ArrayList`.
`clear`	Remove todos os elementos de `ArrayList`.
`contains`	Retorna `true` se `ArrayList` contiver o elemento especificado; do contrário, retorna `false`.
`get`	Retorna o elemento no índice especificado.
`indexOf`	Retorna o índice da primeira ocorrência do elemento especificado em `ArrayList`.
`remove`	Remove a primeira ocorrência do valor especificado.
`remove`	Remove o elemento no índice especificado.
`size`	Retorna o número de elementos armazenados no `ArrayList`.
`trimToSize`	Reduz a capacidade de `ArrayList` de acordo com o número de elementos atual.

Boas práticas de programação

- Um índice precisa ser um valor int ou um valor de um tipo que possa ser promovido a int - byte, short ou char. NÃO pode ser long, caso contrário ocorrerá um erro de compilação.
- ERRO COMUM:

  int [ ] a, b, c → 3 arrays
  int a[ ], b, c → 1 array e dois ints
- Vetores são sempre passados por referência para outros métodos

Introdução a classes e objetos

Introdução

- Durante nossos primeiros programas, definimos os tipos de variáveis de acordo com os dados que queremos usar.
- Muitas vezes precisamos representar os dados do mundo real em nossos programas, no entanto não podemos reduzir uma Pessoa, uma Disciplina, uma Biblioteca em um único tipo, para lidar com isso utilizamos um conjunto de dados com Strings, inteiros, vetores para representar um único objeto.
- Em Prog II - vocês aprendem a utilizar structs que estão fortemente associados a esta representação.
- Em Java (linguagem adotada nesta disciplina) NÃO existem STRUCTS. Ao invés disso, tudo é objeto!

Classe x Struct

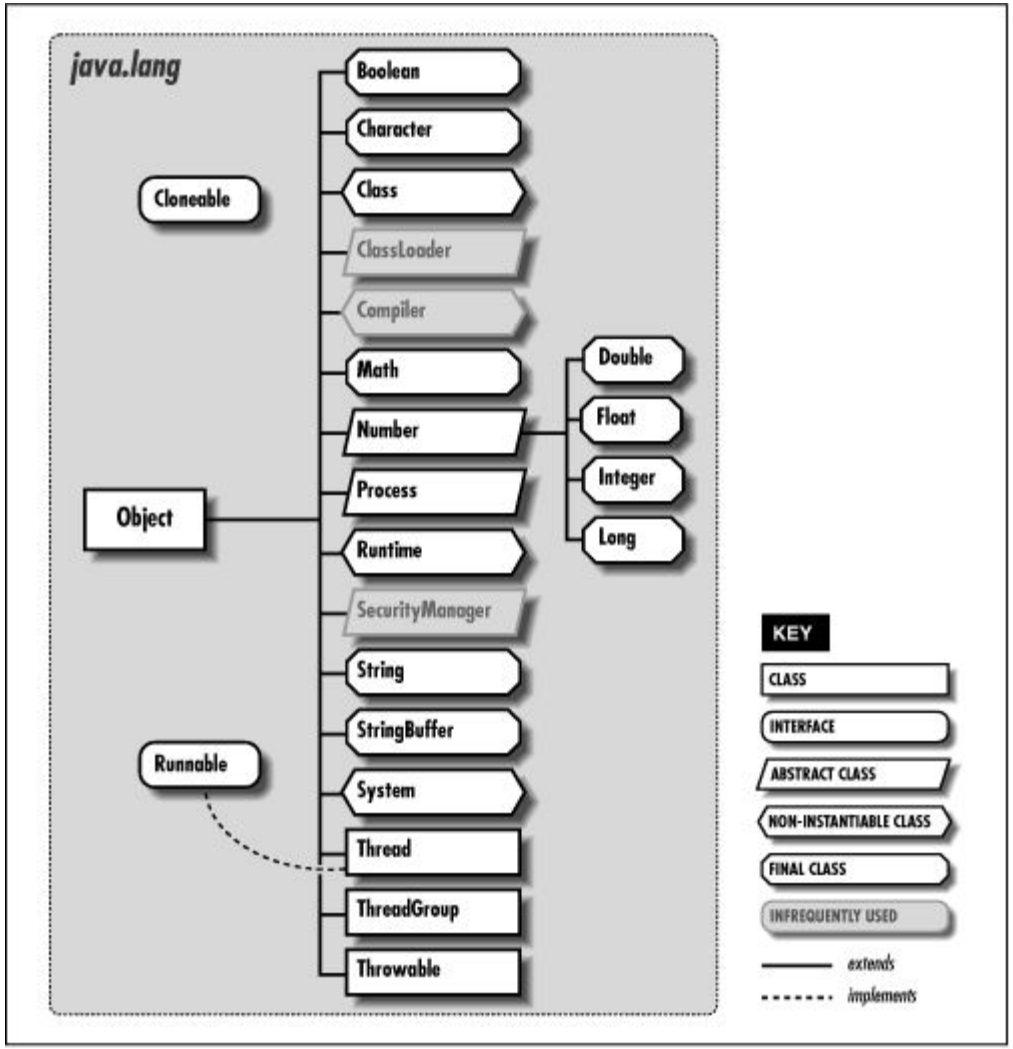
- A única diferença entre classes e structs em C++ é a visibilidade padrão dos membros dessa estrutura.
- Em Structs todos os itens são públicos
- Em Classes todos os itens são, por padrão private.

PS: ISSO EM C++ … JAVA É OUTRA HISTÓRIA

Object

TODAS as classes de objetos herdam as características da classe Object, isto é, toda classe definida em Java é também um Object.

“Filho de peixe…”



Classes

Consideraremos que:

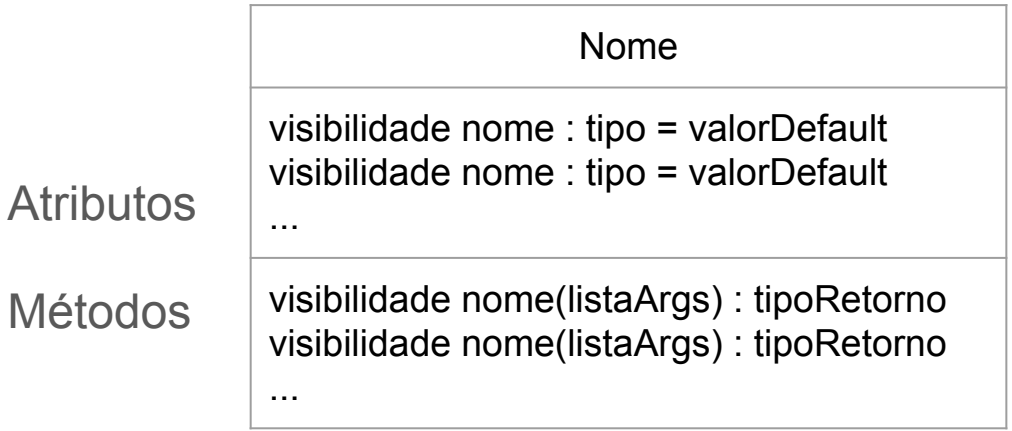
“Uma classe é um gabarito para a definição de objetos. Através da definição de uma classe, descreve-se que propriedades -- ou atributos -- o objeto terá.”

- Além dos atributos também definimos o comportamento desta classe. Isto é feito através de métodos que fazem PARTE da definição da classe.
  - Métodos são funções que fazem parte da descrição de um objeto.
  - Métodos são responsáveis por alterar o estado do objeto, variáveis da classe.

Classes

Orientação a Objetos está fortemente ligado a diversos tipos de técnicas de desenvolvimento e também, é comumente utilizado uma linguagem de modelagem para representar as classes.

Utilizando UML (Unified Modeling Language), representamos uma classe da seguinte maneira:



Classes

Conforme o diagrama UML do slide anterior

Nome da classe: Um identificador para a classe, nomes que façam sentido e que represente claramente o tipo de objeto que será alterado por ela.

Atributos: Conjunto de propriedades da classe, para cada atributo especificamos:

- nome: identificador claro!
- tipo: qual o tipo do atributo, pode ser outra classe.
- valorDefault: valor padrão que deve ser associado a este atributo
- visibilidade: quão acessível este atributo é em relação a outros objetos e outros trechos de código que não a própria classe.

Classes

Métodos: Conjunto de funcionalidades que a nossa classe possui. Assim como os atributos, para cada item definimos:

- nome: identificador claro que será utilizados para chamar o método.
- tipo: caso o método tenha algum tipo de retorno, qual o tipo deste retorno.
- listaArgs: quais os argumentos necessários para este método.
- visibilidade: assim como os atributos, define o quão visível este método é!

VISIBILIDADE

As técnicas de programação orientada a objetos recomendam que a estrutura de um objeto e a implementação de seus métodos devem ser tão privativos como possível. Normalmente, os atributos de um objeto não devem ser visíveis externamente. Da mesma forma, de um método deve ser suficiente conhecer apenas sua especificação, sem necessidade de saber detalhes de como a funcionalidade que ele executa é implementada.

Programando

Class

Lembrem da estrutura inicial do nosso código em Java:

```
public class Main {
	public static void main(String[ ] args) {
	}
}
```

Agora vamos definir uma nova Classe que terá apenas uma responsabilidade, inicialmente nossas classes parecerão muito com structs e terão a função de guardar o estado dos objetos.

HelloWorld em POO

- Classe Lampada

Lampada
private estado : booelan = false
public acender( ) : void public apagar( ) : void public estadoAtual( ) : boolean

Class ContaCorrente

Exemplo de aplicação mais real?

```
public class ContaCorrente {

    //QUAIS OS ATRIBUTOS TEM A CLASSE ContaCorrente?

    //Quais os métodos / ações que podemos utilizar com essa classe?

}
```

PARA TESTAR: UTILIZAREMOS AS CLASSES SEPARADAS EM ARQUIVOS (CÓDIGO FONTES) DIFERENTES! INCLUSIVE A MAIN DA DEFINIÇÃO DE OUTRAS CLASSES.

“Supermodelo”

- Supermodelos são modelos extremamente complexos que tentam se aproximar ao máximo da realidade, uma classe pode ser tão simples quanto conter um atributo ou método, como pode vir a ter muitos atributos e muitos métodos… Essa classe “exagerada” é chamada de supermodelo.

DISCUSSÃO: Que problemas isso pode gerar?

Classes e Objetos

Quem é o dono desta ContaCorrente?

É uma pessoa? Uma pessoa pode ser identificada apenas pelo nome?

Então criaremos uma outra Classe chamada Pessoa que guardará as informações referentes a uma Pessoa e, depois disso associada a ContaCorrente.

UMA CLASSE,
UMA RESPONSABILIDADE

Visibilidade

Utilizamos as palavras-chave a seguir para determinar a visibilidade de cada atributo e método da nossa classe.

- public - externamente visível
- private - visível apenas internamente, pela própria classe.
- protected - apenas classes que herdam podem alterar estes atributos e utilizar os métodos.

Atenção: Em JAVA, quando não explicitamos a visibilidade, o atributo / método é package-public, isto é, definições de classes que estejam no mesmo pacote (pasta) podem acessar e alterar os valores.

Leitura Recomendada

Livro base:

- Capítulos 3, 6 e 8.

Online

- https://en.wikipedia.org/wiki/Object-oriented_programming
- http://www.dca.fee.unicamp.br/cursos/PooJava/classes/conceito.html
- https://www.devmedia.com.br/abstracao-encapsulamento-e-heranca-pilares-da-poo-em-java/26366

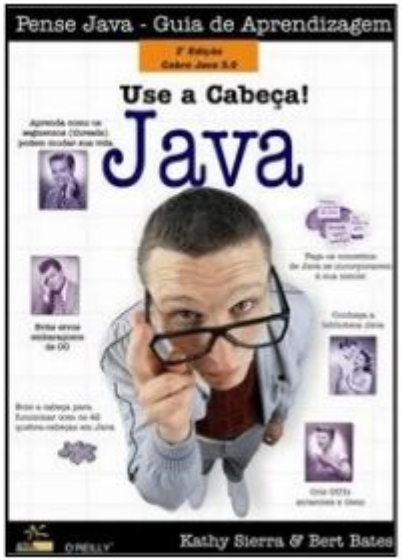
Referências

Livro

Java: Como programar.

Use a Cabeça! Java

Online



Contato

vinicius.machado@osorio.ifrs.edu.br