Conteúdo

1. Introdução
2. Listas
3. Pilhas e Filas
4. Árvores
5. Árvores de Pesquisa
   - Árvore Binária e Árvore AVL
   - Árvore N-ária e Árvore B
6. Tabelas de Dispersão (Hashing)
7. Métodos de Acesso a Arquivos
8. Métodos de Ordenação de Dados

Alocação Dinâmica de Memória

Objetos

Criação de Objetos

Em Java, um objeto novo de uma classe é criado usando-se o operador **new**.

O operador **new** cria um novo objeto a partir de uma classe especificada e retorna uma ***referência*** para este objeto.

Exemplos:

- Pessoa p = new Pessoa (“Maria”);
- Integer[ ] inteiros = new Integer[10];

Objetos

Criação de Objetos

A chamada do operador **new** resulta em:

- Um novo objeto é dinamicamente alocado na memória, e todas as variáveis de instância (atributos) são inicializadas com seus valores padrão (null, 0 ou false).
- O construtor para o novo objeto é chamado com os parâmetros especificados.
- Depois do construtor retornar, o operador **new** retorna uma referência (isto é, um endereço de memória) para o novo objeto recém criado.

Objetos

Criação de Objetos

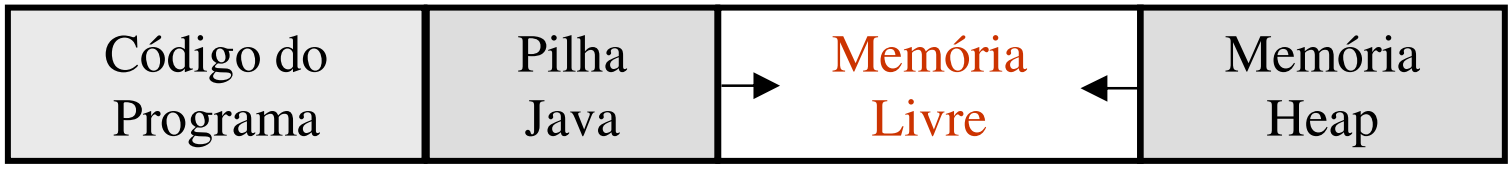
- Uma variável referência pode ser entendida como sendo um "ponteiro" para um objeto.
- Toda variável referência deve apontar para algum objeto, a menos que seja **null**, caso em que não aponta para nada.
- Pode haver várias referências para um mesmo objeto, e cada referência para um objeto específico pode ser usada para acessar os métodos e as variáveis de instância associadas com aquele objeto.
- Se uma variável referência for usada para alterar o estado do objeto, esta mudança será visível para todas as outras referências.

Gerenciamento de Memória

➡ A memória do computador é simplesmente uma seqüência de *palavras* da memória, e cada qual consiste em 8, 16, ou mais bytes (dependendo do computador).

- A memória em um computador pode ser visualizada como basicamente um array gigante de palavras de memória.
- As palavras da memória são enumeradas de 0 a N-1, onde N é o número de palavras de memória disponíveis no computador.
- O número associado com cada espaço na memória do computador é conhecido como **endereço**.

Gerenciamento de Memória



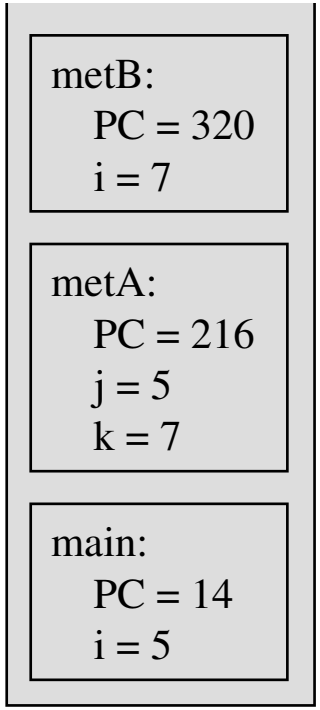
Código do Programa: tamanho fixo (não aumenta). Seqüência de códigos byte que são definidos como instruções de "máquina".

Pilha Java: cresce para uma memória maior. Aloca espaço para as variáveis locais e outras informações importantes para cada método (valores correntes das variáveis locais, parâmetros do método, informações do método que o chamou e o que precisa ser retornado pelo método).

Memória Heap: cresce para uma memória menor. Memória para os objetos alocada dinamicamente durante a execução dos métodos, através do operador *new*.

Gerenciamento de Memória

Exemplo:



Pilha Java

```
      main () {
        int i = 5;
        …
14      metA (i);
       …
      }

      metA (int j){
        int k = 7;
        …
216    metB (K);
        …
       }

320 metB (int m){
        …
       }
```

Programa Java

Variáveis Estáticas e Dinâmicas

- Dados são acessados através de variáveis.

Variáveis estáticas: são variáveis cujo espaço em memória é reservado quando o programa ou bloco de código começa a ser executado e liberado quando o mesmo termina. A reserva e liberação da memória ocorre de forma automática.

Variáveis dinâmicas: são variáveis cujo espaço em memória contém o endereço de um bloco de memória alocado de forma explícita durante o decorrer do programa.

- No Java, uma variável é uma referência a um objeto (variável dinâmica) ou é de um tipo primitivo (variável estática).

Gerenciamento de Memória

⇨ Todos objetos consomem recursos do sistema e é importante liberar estes recursos quando um objeto não é mais necessário.

⇨ Algumas linguagens de programação (por exemplo, C e C++) requerem a destruição explícita dos objetos.

⇨ Em Java, o gerenciamento de memória é de responsabilidade do ambiente de execução.

De vez em quando, a JVM (Java Virtual Machine) pode notificar que o espaço disponível na memória heap está se tornando escasso e decidir por recuperar o espaço que está sendo usado por objetos que não são mais referenciados. Este processo de reparação é conhecido como coleta de lixo.

Gerenciamento de Memória

- No Java, um objeto pára de existir quando não existem mais referências a ele e ele é destruído pelo coletor de lixo, liberando a memória correspondente.
- O programador Java não sabe quando o coletor de lixo será automaticamente invocado e quando um objeto tem seus recursos liberados.

⇨ É importante liberar todas as referências a um objeto quando ele não é mais necessário.

Se alguma referência a um objeto é mantida, ele não será coletado pelo coletor de lixo.

Alocação Dinâmica de Memória

```
public static void main(String[] args) {
  StringBuffer sb = new StringBuffer(200);
  Runtime.getRuntime().gc();
  sb.append("antes da alocacao ").append(Runtime.getRuntime().freeMemory());
  System.out.println(sb);
  byte[] ba = new byte[102400];
  sb.setLength(0);
  sb.append("apos a alocacao ").append(Runtime.getRuntime().freeMemory());
  System.out.println(sb);
  System.out.println(ba.length);
  ba = null; // libera referencia
  Runtime.getRuntime().gc();
  sb.setLength(0);
  sb.append("apos a liberacao ").append(Runtime.getRuntime().freeMemory());
  System.out.println(sb);
}
```

Saída 1:

```
  antes da alocacao 1924320
  apos a alocacao 1811360
  102400
  apos a liberacao 1924320
```

Alocação Dinâmica de Memória

```
public static void main(String[] args) {
  StringBuffer sb = new StringBuffer(200);
  Runtime.getRuntime().gc();
  byte[] ba = new byte[102400];
  sb.append("apos a alocacao ").append(Runtime.getRuntime().freeMemory());
  System.out.println(sb);
  System.out.println(ba.length);
  ba = null; // libera referencia
  Runtime.getRuntime().gc();
  sb.setLength(0);
  sb.append("apos a liberacao ").append(Runtime.getRuntime().freeMemory());
  System.out.println(sb);
  try {
    System.out.println(ba.length);
  } catch (NullPointerException e) {
    System.out.println("gerou excecao " + e.getClass());  }
  }
```

Saída 2:

```
  apos a alocacao 1811416
  102400
  apos a liberacao 1924320
  gerou excecao class java.lang.NullPointerException
```

Exercícios

- Atribuição de valores em variáveis estáticas:

```
int a, b;
a = 1;
b = a;
a = 2;
```

Valor de b?

Exercícios

- Atribuição de valores em variáveis dinâmicas:

```
Circulo c1, c2;
c1 = new Circulo (0.5); // raio
c2 = c1;

c1.setRaio (1.0);
int r1 = c1.getRaio();
int r2 = c2.getRaio();
```

Quais os valores de r1 e r2?

Exercícios

```
public static void metA (Integer a, int b){
        a = 20;
        b = 50;
}

public static void main(String[] args) {
        Integer a = 10;
        int b = 5;
        metA (a, b);
        System.out.println(a);
        System.out.println(b);
}
```

Exercícios

```
public static void metB (StringBuffer sb1){
        sb1.setLength(0);
        sb1.append("girafa");
        sb1 = new StringBuffer(“leao");
}

public static void main(String[] args) {
        StringBuffer sb1 = new StringBuffer ("macaco");
        metB (sb1);
        System.out.println(sb1);
}
```

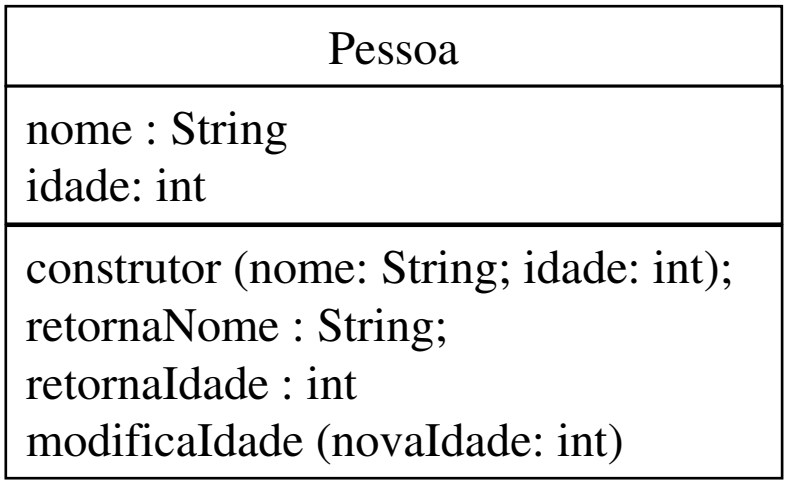
Exercícios

```
public static void metC (StringBuffer sb2){
        sb2 = new StringBuffer ();
        sb2.append("leao");
}

public static void main(String[] args) {
        StringBuffer sb2 = new StringBuffer ("cachorro");
        metC (sb2);
        System.out.println(sb2);
}
```

Exercícios

- Considere a classe Pessoa:



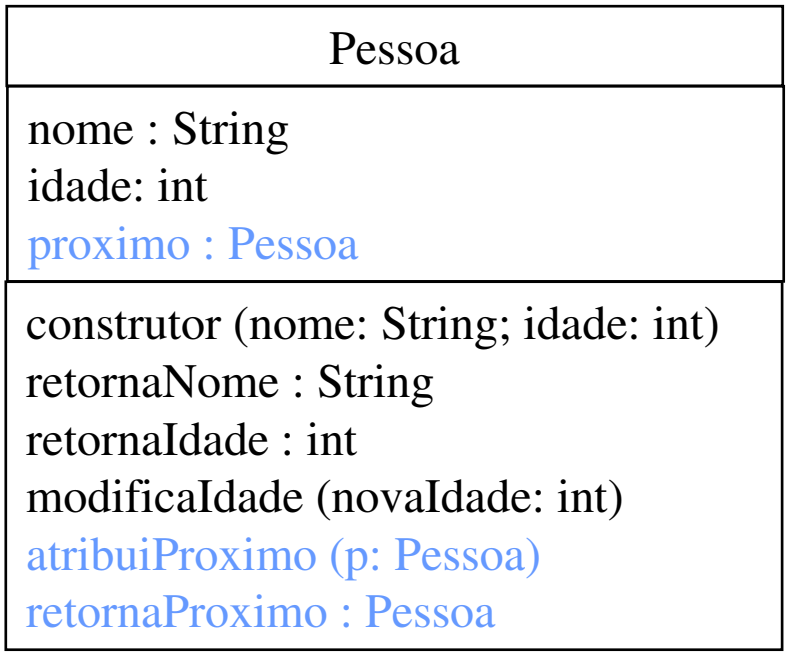
Exercícios

```
public static void main (String[] args){
  Pessoa pessoaA,pessoaB,pessoaC;
  pessoaA = new Pessoa("Maria",20);
  pessoaC = new Pessoa("Jose",20);
  pessoaB = pessoaA;
  pessoaB.idade = 21;
  pessoaC = new Pessoa("Antonio", 22);

  pessoaA = new Pessoa ("Jose", 20);
  pessoaA.idade = 30;
  Pessoa temp = pessoaC;
  pessoaC = pessoaA;
  pessoaC.idade = 25;
  pessoaA = pessoaB;
  pessoaB = temp;
  pessoaA.idade = pessoaC.idade;
}
```

Exercícios

- Considere a classe Pessoa:



Exercícios

```
public static void main (String[] args){
  Pessoa pessoaA,pessoaB,pessoaC;
  pessoaA = new Pessoa("Maria",20);
  pessoaC = new Pessoa("Jose",20);
  pessoaB = pessoaA;
  pessoaB.idade = 21;
  pessoaC = new Pessoa("Antonio", 22);
  pessoaA = new Pessoa("Mariana",18);
  pessoaA.proximo = pessoaB;
  pessoaB.proximo = pessoaC;

  pessoaB = new Pessoa ("Jose", 20);
  pessoaB.proximo = pessoaC;
  Pessoa temp = pessoaA.proximo;
  temp.proximo = pessoaB;
  temp = new Pessoa ("Marta", 40);
  pessoaC.proximo = temp;
}
```