Matéria: Programação

Assunto: Estruturas de Dados, Algoritmos e Programação Orientada a Objetos em Java

---

Resumo Teórico do Assunto

Para resolver as questões apresentadas, é fundamental compreender os conceitos de **Estruturas de Dados**, **Algoritmos de Busca** e o **Ciclo de Vida de Objetos em Java**, especialmente a ordem de inicialização.

I. Estruturas de Dados e Algoritmos de Busca

As estruturas de dados são formas de organizar e armazenar dados para que possam ser acessados e manipulados de forma eficiente. Os algoritmos de busca são procedimentos para encontrar um item específico dentro de uma estrutura de dados.

 # 1. Estruturas de Dados Fundamentais

• Array (Vetor):
* Uma coleção de elementos do mesmo tipo, armazenados em posições de memória **contíguas**.
* Possui um tamanho fixo (estático) após a criação.
* Permite **acesso direto** (ou aleatório) a qualquer elemento usando seu **índice**, com complexidade de tempo **O(1)**. Isso significa que o tempo para acessar um elemento é constante, independentemente do tamanho do array.
* Exemplo: `int[] numeros = new int[1000];`

• Lista Encadeada (Linked List):
* Uma coleção de **nós**, onde cada nó contém um dado e uma **referência (ponteiro)** para o próximo nó na sequência.
* A alocação de memória é **dinâmica**, ou seja, os nós podem ser espalhados pela memória e não precisam ser contíguos.
* O acesso aos elementos é **sequencial**, exigindo que se percorra a lista desde o início até o nó desejado. A complexidade de tempo para acessar um elemento arbitrário é **O(n)**, onde 'n' é o número de elementos, pois no pior caso, é preciso percorrer todos os nós.
* Exemplo: `Node primeiro;` (onde `Node` é uma classe com `data` e `next`).

 # 2. Algoritmos de Busca

• **Busca Sequencial (Linear Search):**
* **Como funciona:** Percorre a coleção elemento por elemento, do início ao fim, comparando cada item com o valor procurado.
* **Quando usar:** Funciona em coleções **ordenadas ou desordenadas**. É a única opção para listas encadeadas desordenadas ou quando o acesso direto não é possível.
* **Eficiência:** Complexidade de tempo **O(n)** no pior caso, pois pode ser necessário verificar todos os 'n' elementos.

• **Busca Binária (Binary Search):**
* **Como funciona:** Algoritmo muito eficiente que exige que a coleção esteja **ordenada**. Funciona dividindo repetidamente a parte da lista que pode conter o item ao meio. Compara o valor procurado com o elemento do meio:
* Se forem iguais, o item é encontrado.
* Se o valor procurado for menor, a busca continua na metade inferior da coleção.
* Se o valor procurado for maior, a busca continua na metade superior da coleção.
* **Quando usar:**
* **Obrigatório:** A coleção deve estar **ordenada**.
* **Ideal:** Funciona melhor em estruturas que permitem **acesso direto** ao meio em O(1), como **arrays**.
* **Não eficiente para Listas Encadeadas:** Embora uma lista encadeada possa estar ordenada, a busca binária não é eficiente nela porque encontrar o "meio" da lista ainda exige uma travessia sequencial (O(n)), anulando a vantagem logarítmica. Portanto, para listas encadeadas (mesmo que ordenadas), a busca sequencial é geralmente a abordagem prática e mais eficiente em termos de implementação e custo real.
* **Eficiência:** Complexidade de tempo **O(log n)**, tornando-a muito mais rápida que a busca sequencial para grandes coleções.

II. Programação Orientada a Objetos em Java: Classes, Objetos e Ciclo de Vida

A Programação Orientada a Objetos (POO) em Java baseia-se em conceitos como classes, objetos, variáveis de instância, métodos, construtores e blocos de inicialização. A ordem em que esses elementos são processados durante a criação de um objeto é crucial.

 # 1. Conceitos Fundamentais

• **Classe:** Um **molde** ou **projeto** para criar objetos. Define a estrutura (variáveis) e o comportamento (métodos) que os objetos dessa classe terão.
* Exemplo: `public class Tst { ... }`

• **Objeto (Instância):** Uma **instância concreta** de uma classe. Cada objeto possui seu próprio conjunto de **variáveis de instância** (estado) e pode executar os **métodos** definidos na classe.

* Exemplo: `new Tst(4, -4)` cria um novo objeto da classe `Tst`.

• **Variáveis de Instância (Atributos):** Variáveis declaradas dentro de uma classe, mas fora de qualquer método ou construtor. Elas definem o **estado** de um objeto. Cada objeto tem sua própria cópia dessas variáveis.
* Exemplo: `int ini=0, fim=25;`

• **Métodos:** Funções que definem o **comportamento** de um objeto.
* Exemplo: `void print() { ... }`

• **Construtor:** Um tipo especial de método usado para **inicializar** um objeto no momento de sua criação (`new`).
* Tem o **mesmo nome da classe**.
* **Não possui tipo de retorno** (nem mesmo `void`).
* Pode receber parâmetros para personalizar a inicialização.
* Exemplo: `Tst(int a, int b) { ... }`

• Bloco de Inicialização de Instância (IIB - Instance Initialization Block):
* Um bloco de código `{ ... }` que não pertence a nenhum método ou construtor.
* É executado **toda vez que um novo objeto é criado**, *antes* do construtor.
* Se houver múltiplos IIBs, eles são executados na **ordem em que aparecem** no código da classe.
* Exemplo: `{ ini=fim%7; fim=ini*3; }`

• Método `main`:
* `public static void main(String[] args)` é o **ponto de entrada** de um programa Java.
* É `static` porque pode ser chamado sem a necessidade de criar um objeto da classe `Main`.

 # 2. Ordem de Execução na Criação de um Objeto em Java

Esta é a sequência **fundamental** para entender como as variáveis de instância são inicializadas e modificadas quando um novo objeto é criado:

1. **Inicialização Padrão:** Todas as variáveis de instância são inicializadas com seus valores padrão (0 para tipos numéricos, `null` para referências de objeto, `false` para booleanos).
2. **Inicialização Explícita:** As variáveis de instância são inicializadas com os valores definidos diretamente na sua declaração na classe (ex: `int ini=0, fim=25;`).
3. **Blocos de Inicialização de Instância (IIB):** Todos os blocos de inicialização de instância são executados, na **ordem em que aparecem** no código da classe.
4. **Construtor:** O construtor chamado (aquele que corresponde aos argumentos passados com `new`) é executado.

Exemplo de Fluxo (para a Questão 54):

Considerando a classe `Tst`:

```java
public class Tst {
int ini=0,fim=25; // (A) Inicialização Explícita

{ // (B) Primeiro Bloco de Inicialização de Instância
ini=fim%7;
fim=ini*3;
}

Tst(int a, int b) { // (D) Construtor
ini+=a;
fim+=b;
}

{ // (C) Segundo Bloco de Inicialização de Instância
ini/=2;
fim+=10;
}

void print() {
System.out.println(ini+fim);
}
}
```

Quando `new Tst(4, -4)` é executado:

1. **Inicialização Padrão:** `ini` e `fim` seriam 0 (se não houvesse inicialização explícita).
2. **Inicialização Explícita (A):** `ini` se torna `0`, `fim` se torna `25`.
3. **Primeiro IIB (B):**
* `ini = fim % 7` -> `ini = 25 % 7 = 4`
* `fim = ini * 3` -> `fim = 4 * 3 = 12`
* Estado atual: `ini = 4`, `fim = 12`
4. **Segundo IIB (C):**
* `ini /= 2` -> `ini = 4 / 2 = 2`
* `fim += 10` -> `fim = 12 + 10 = 22`
* Estado atual: `ini = 2`, `fim = 22`
5. **Construtor (D) `Tst(4, -4)`:**
* `ini += a` -> `ini = 2 + 4 = 6`
* `fim += b` -> `fim = 22 + (-4) = 18`
* Estado final do objeto: `ini = 6`, `fim = 18`

Finalmente, `print()` exibirá `ini + fim`, ou seja, `6 + 18 = 24`.

Compreender esses conceitos e a ordem de execução é essencial para analisar e prever o comportamento de programas Java e para escolher os algoritmos mais eficientes para diferentes cenários de dados.

Questões de Provas Anteriores

Fonte: escriturario_agente_de_tecnologia (1).pdf, Página: 20

pcimarkpci MjgwNDowMTRkOjE0YTU6OTI1ODozOGQ2OjNhMGM6NTM0MzplZmI1:U3VuLCAyNyBKdWwgMjAyNSAyMzo0Nzo0MCAtMDMwMA==
www.pciconcursos.com.br
20
BANCO DO BRASIL
AGENTE DE TECNOLOGIA - Microrregião 16 DF-TI GABARITO 1
53
Desejam-se realizar buscas nas seguintes coleções de dados, representadas na linguagem Java:
I - Um array de 1.000 números inteiros ordenados de forma decrescente;
II - Uma lista encadeada desordenada e alocada dinamicamente, cujos 1.000 nós contêm strings (uma string por nó);
III - Uma lista encadeada, alocada dinamicamente, cujos 1.000 nós contêm números decimais (um número double por nó)
ordenados de forma ascendente.
Levando-se em consideração a exequibilidade e a eficiência, quais métodos de busca devem ser empregados, respectiva-
mente, em cada um dos três casos acima?

(A) I – sequencial; II – sequencial; III – binária
(B) I – binária; II – sequencial; III – sequencial
(C) I – binária; II – sequencial; III – binária
(D) I – sequencial; II – sequencial; III – sequencial
(E) I – sequencial; II – binária; III – binária
54
As classes Java a seguir são públicas e ocupam arquivos separados.

```
public class Tst {

int ini=0,fim=25;

void print() {
```

```
System.out.println(ini+fim);
}

{
ini=fim%7;
fim=ini*3;
}

Tst(int a, int b) {
ini+=a;
fim+=b;
}

{
ini/=2;
fim+=10;
}
}
public class Main {
public static void main(String[] args) {
new Tst(4, -4).print();
}
}
```

O que será exibido no console quando o método main for executado?

(A) 0
(B) 10
(C) 24
(D) 25
(E) 33