Matéria: Tecnologia da Informação

Assunto: Programação e Algoritmos

Resumo Teórico do Assunto

Para dominar as questões de Programação e Algoritmos, especialmente em concursos, é fundamental compreender os conceitos de **Programação Orientada a Objetos (POO)**, **Estruturas de Dados** e **Análise de Algoritmos**.

---

1. Programação Orientada a Objetos (POO) e Sintaxe de Linguagens (Java e Kotlin)

A Programação Orientada a Objetos é um paradigma de programação que organiza o software em torno de **objetos**, em vez de funções e lógica.

• **Classe:** Uma **classe** é um **molde** ou **planta** para criar objetos. Ela define as características (atributos) e os comportamentos (métodos) que os objetos criados a partir dela terão.
* Exemplo: `AlunoJava` ou `AlunoKotlin` são classes que definem o que um "aluno" é no sistema.
• **Objeto (Instância):** Um **objeto** é uma **instância** concreta de uma classe. É a materialização da planta.
• **Atributos (Propriedades/Campos):** São as **variáveis** que definem o **estado** de um objeto. Cada objeto terá seus próprios valores para esses atributos.
* Exemplo: `codigo`, `nome`, `numero`, `texto` são atributos da classe `AlunoJava`.
• **Modificadores de Acesso:** Controlam a **visibilidade** e o acesso aos atributos e métodos de uma classe.
* **`public`**: O membro é acessível de qualquer lugar.
* **`private`**: O membro é acessível apenas dentro da própria classe. Isso promove o **encapsulamento**, um pilar da POO, que protege os dados internos do objeto.
• **Construtor:** É um **método especial** usado para **inicializar** um novo objeto quando ele é criado. Ele tem o mesmo nome da classe e não possui tipo de retorno.
* Exemplo: `public AlunoJava (String codigo, String nome)` é um construtor que recebe código e nome para inicializar um objeto `AlunoJava`.
• **`this` Keyword:** Em Java, `this` é uma referência ao **objeto atual** (a instância da classe em que o código está sendo executado). É frequentemente usado para distinguir entre um atributo da classe e um parâmetro de método com o mesmo nome (ex: `this.codigo = codigo`).

 # Diferenças Sintáticas Chave entre Java e Kotlin (relevantes para a questão):

Kotlin é uma linguagem moderna que visa ser mais concisa e expressiva que Java, mantendo

a interoperabilidade.

• **Declaração de Classe:**
* **Java:** `public class NomeClasse { ... }`
* **Kotlin:** `class NomeClasse { ... }` (o `public` é o padrão e pode ser omitido).
• **Declaração de Propriedades (Atributos):**
* **Java:** `private String nome;` (tipo vem depois do modificador e antes do nome).
* **Kotlin:** `private var nome: String` (ou `private val nome: String`).
* **`var`**: Variável mutável (pode ter seu valor alterado).
* **`val`**: Variável imutável (somente leitura, seu valor não pode ser alterado após a inicialização).
* O tipo (`: String`) vem após o nome da variável.
• **Construtor Primário em Kotlin:** Kotlin permite definir um **construtor primário** diretamente na declaração da classe, tornando-a muito mais concisa. As propriedades declaradas no construtor primário podem ser automaticamente inicializadas e se tornam atributos da classe.
* Exemplo: `class AlunoKotlin (val nome: String, val codigo: String)` declara uma classe `AlunoKotlin` com um construtor que recebe `nome` e `codigo` e os define como propriedades imutáveis.
• **Inicialização de Propriedades:** Em Kotlin, propriedades podem ter valores padrão diretamente na declaração (`var numero = 0`).

---

2. Algoritmos e Estruturas de Dados

Algoritmo: Uma sequência finita e bem definida de instruções para resolver um problema ou realizar uma tarefa.

 # Estruturas de Dados: Vetores (Arrays)

• **Vetor (Array):** É uma **estrutura de dados linear** que armazena uma coleção de elementos do mesmo tipo em posições de memória contíguas. Cada elemento é acessado por um **índice** numérico.
* **Características:** Tamanho fixo (geralmente), acesso direto (O(1)) a qualquer elemento pelo índice.
• **Ordenação:** O processo de organizar os elementos de uma estrutura de dados em uma sequência específica (crescente ou decrescente). A ordenação é um **pré-requisito fundamental** para a aplicação de certos algoritmos de busca, como a Busca Binária.

 # Algoritmos de Busca: Busca Binária

• **Busca Binária (Binary Search):** É um algoritmo de busca **altamente eficiente** para encontrar um item em uma **lista (vetor) *ordenada***.
* **Como funciona:** O algoritmo compara o item procurado com o elemento do meio da lista.
* Se forem iguais, o item é encontrado.

* Se o item procurado for menor, a busca continua na metade esquerda da lista.
* Se o item procurado for maior, a busca continua na metade direita da lista.
* Esse processo de divisão pela metade é repetido até que o item seja encontrado ou o subvetor de busca se torne vazio.
* **Pré-requisito:** A lista (vetor) **DEVE ESTAR ORDENADA**. Se não estiver, a busca binária não funcionará corretamente.

 # Análise de Algoritmos: Complexidade de Tempo

• **Complexidade de Tempo:** É uma medida de como o **tempo de execução** de um algoritmo cresce à medida que o **tamanho da entrada (N)** aumenta. É expressa usando a **Notação Big O (O())**, que descreve o comportamento assintótico (para grandes valores de N).
• **Pior Caso (Worst Case):** Refere-se ao cenário em que o algoritmo leva o **tempo máximo** possível para ser executado. É a métrica mais comum para avaliar a eficiência de um algoritmo em concursos, pois garante que o algoritmo nunca será mais lento do que o tempo especificado.
• **Complexidade da Busca Binária:**
* No **pior caso**, a Busca Binária divide o problema pela metade a cada iteração. Isso leva a uma complexidade de tempo **logarítmica**, denotada como **O(log N)**.
* Para um vetor de `N` elementos, o número máximo de comparações (iterações) necessárias para encontrar um elemento (ou determinar que ele não existe) é aproximadamente `log■N`.
* **Fórmula para o número de iterações no pior caso:** `log■N + 1` (ou `■log■N■ + 1` ou `■log■N■`). A variação exata depende da definição de "iteração" e se o elemento é encontrado ou não.
* **`log■N`**: Logaritmo de N na base 2.
* **`■x■` (Função Piso):** O maior inteiro menor ou igual a `x`. Ex: `■3.7■ = 3`, `■5■ = 5`.
* **`■x■` (Função Teto):** O menor inteiro maior ou igual a `x`. Ex: `■3.7■ = 4`, `■5■ = 5`.

Compreender esses conceitos permitirá analisar e comparar soluções de programação, bem como prever o desempenho de algoritmos em diferentes cenários.

Questões de Provas Anteriores

Fonte: escriturario_agente_de_tecnologia (1).pdf, Página: 23

pcimarkpci MjgwNDowMTRkOjE0YTU6OTI1ODozOGQ2OjNhMGM6NTM0MzplZmI1:U3VuLCAyNyBKdWwgMjAyNSAyMzo0Nzo0MCAtMDMwMA==

BANCO DO BRASIL
AGENTE DE TECNOLOGIA - Microrregião 16 DF-TIGABARITO 1

61
Foi solicitado a um programador de sistemas de informação que transformasse uma classe escrita em Java em uma classe
equivalente, para ser executada em um programa Kotlin.
O código da classe Java é:

```
public class AlunoJava{
private String codigo;
private String nome;
private int numero=0;
private String texto= “EscolaX”;
public AlunoJava (String codigo,String nome)
{ this.codigo = codigo;
this.nome = nome; }
}
```

A classe em Kotlin equivalente à classe Java acima é

(A) public class AlunoKotlin (private String: nome , private String: codigo)
{ private:
numero int = 0
texto String = “EscolaX” }
(B) public class AlunoKotlin (private var nome; codigo: String)
{ private var numero = 0
private var texto = “EscolaX” }
(C) class AlunoKotlin (val nome: String, val codigo: String)
{ private this.nome = nome
private this.codigo=codigo
private var int numero = 0
private var String texto = “EscolaX” }
(D) class AlunoKotlin (var nome: String, var codigo: String)
{ private var numero = 0
private var texto = “EscolaX”
private AlunoKotlin.nome, AlunoKotlin.codigo }
(E) class AlunoKotlin (private val nome: String, private val codigo: String)
{ private var numero = 0
private var texto = “EscolaX” }
62
O gerente de uma agência bancária recebe, diariamente, solicitações de seus clientes com dúvidas sobre a melhor de-
cisão para aplicações financeiras e as armazena, com um código numérico crescente, num vetor de solicitações, para
respondê-las ao final do expediente. Para manter o conceito de bom atendimento, o gerente gostaria, sempre que pos-
sível, que a ordem das respostas seguisse, estritamente, a ordem de chegada das

solicitações. Entretanto, há casos em
que é necessário, por motivos de emergência ou por prioridade legal, localizar determinado código numérico para atender
à solicitação correspondente antes das demais, “furando” a fila de espera. O gerente solicitou, então, à equipe de TI do
banco, uma proposta que conciliasse essas duas necessidades. Ao estudar o problema, a equipe de TI concluiu que uma
solução que mapearia diretamente essa necessidade da gerência seria permitir a realização de uma busca binária sobre
o vetor de solicitações ordenado pelos seus códigos numéricos.
Verificando a viabilidade dessa sugestão, o grupo de TI calculou que, se considerar a existência de N solicitações, a quan-
tidade de iterações necessárias para localizar determinado código numérico no vetor de solitações, utilizando a busca
binária, no pior caso, é
(A)
2log N■■■■ , em que a notação x■■■■ significa maior inteiro menor ou igual a x
(B) 1 + 2log N■■■■ , em que a notação x■■■■ significa maior inteiro menor ou igual a x
(C) 1 + 2log N■■■■ , em que a notação x■■■■ significa menor inteiro maior ou igual a x
(D) 2N
(E) 2N■1