Matéria: Programação

Assunto: Java

Resumo Teórico do Assunto

Para resolver questões de Java como a apresentada, é fundamental compreender os conceitos de manipulação de strings e caracteres, estruturas de controle de fluxo e a definição de métodos.

Aqui está um resumo didático dos pontos essenciais:

1. Fundamentos de Java e Tipos de Dados Essenciais

• **Java**: É uma linguagem de programação orientada a objetos, conhecida por sua portabilidade ("write once, run anywhere").
• **String**:
* Representa uma sequência de caracteres. Em Java, `String` é uma **classe**, não um tipo primitivo.
* **Imutabilidade**: Objetos `String` são **imutáveis**. Isso significa que, uma vez criados, seu conteúdo não pode ser alterado. Qualquer operação que pareça modificar uma `String` (como concatenação ou conversão de caixa) na verdade cria uma **nova** `String`.
* **Métodos Comuns**:
* `str.length()`: Retorna o número de caracteres na string.
* `str.charAt(int index)`: Retorna o caractere no índice especificado (índices começam em 0). **Importante**: Não se acessa caracteres de uma `String` em Java usando `str[i]` como em algumas outras linguagens; `charAt()` é o método correto.
* `str.concat(String otherString)`: Concatena a string atual com `otherString`, retornando uma **nova** string.
* **Cuidado**: Se a string original (`str` neste caso) for `null`, chamar `concat()` nela resultará em um `NullPointerException`. É crucial inicializar strings que serão concatenadas (ex: `String r = "";`). O operador `+` também pode ser usado para concatenação e é geralmente mais flexível.
• **char**:
* É um tipo de dado primitivo em Java que representa um único caractere Unicode (letras, números, símbolos).

2. Manipulação de Caracteres com a Classe `Character`

A classe `java.lang.Character` fornece métodos utilitários para trabalhar com caracteres individuais.

• **`Character.isLowerCase(char ch)`**: Retorna `true` se o caractere `ch` for uma letra

minúscula, `false` caso contrário.
• **`Character.toUpperCase(char ch)`**: Converte o caractere `ch` para sua versão maiúscula, se for uma letra. Se não for uma letra ou já for maiúscula, retorna o próprio caractere.
• **`Character.toString(char ch)`**: Converte um caractere `ch` em uma `String` de um único caractere. Isso é útil quando você precisa concatenar um `char` com uma `String` usando o método `concat()`.

3. Estruturas de Controle de Fluxo

São usadas para controlar a ordem de execução das instruções.

• **Laços de Repetição (Loops)**:
* **`while` loop**:
```java
while (condicao) {
// código a ser executado enquanto a condição for verdadeira
}
```
A condição é avaliada **antes** de cada iteração. Se a condição for falsa desde o início, o bloco de código nunca será executado.
* **`do-while` loop**:
```java
do {
// código a ser executado
} while (condicao);
```
O bloco de código é executado **pelo menos uma vez**, e a condição é avaliada **após** cada iteração. Se a condição for verdadeira, o loop continua; caso contrário, ele termina.
• **Condicionais (`if-else`)**:
```java
if (condicao) {
// código se a condição for verdadeira
} else {
// código se a condição for falsa
}
```
Permite que o programa tome decisões com base em uma condição.

4. Definição de Métodos em Java

Métodos são blocos de código que realizam uma tarefa específica.

• **Sintaxe Básica**:
```java
modificador_acesso static tipo_retorno nome_metodo(tipo_parametro nome_parametro, ...) {
```

// corpo do método
return valor_retorno; // se o tipo_retorno não for void
}
```

• **`public`**: Um **modificador de acesso** que indica que o método pode ser acessado de qualquer lugar.
• **`static`**: Indica que o método pertence à classe em si, e não a uma instância específica da classe. Pode ser chamado diretamente usando o nome da classe (ex: `MinhaClasse.meuMetodo()`) sem precisar criar um objeto.
• **`String` (tipo de retorno)**: Especifica o tipo de dado que o método retornará. Se o método não retornar nada, o tipo de retorno é `void`.
• **`converte` (nome do método)**: O nome que identifica o método.
• **`(String str)` (parâmetros)**: A lista de variáveis que o método espera receber como entrada. `str` é o nome da variável que receberá a string passada para o método.
• **`return r;`**: A instrução `return` é usada para enviar um valor de volta ao chamador do método. O tipo do valor retornado deve corresponder ao `tipo_retorno` declarado no método.

**5. Boas Práticas e Erros Comuns**

• **Inicialização de Variáveis**: Sempre inicialize variáveis, especialmente `String`s que serão usadas em concatenações. `String r = "";` (string vazia) é uma inicialização segura para evitar `NullPointerException` ao usar `r.concat(...)`. `String r = null;` sem uma atribuição posterior antes do uso de `concat()` causará um erro.
• **Acesso a Caracteres**: Lembre-se que `str.charAt(i)` é a forma correta de acessar um caractere em uma `String` em Java, não `str[i]`.

Compreendendo esses conceitos, você estará bem preparado para analisar e entender o funcionamento de códigos Java que envolvem manipulação de texto e estruturas de controle.

---

## Questões de Provas Anteriores

**Fonte: escriturario_agente_de_tecnologia.pdf, Página: 26**

**pcimarkpci MjgwNDowMTRkOjE0YTU6OTI1ODozOGQ2OjNhMGM6NTM0MzplZmI1:U3VuLCAyNyBKdWwgMjAyNSAyMzo0NzozMSAtMDMwMA==**
**www.pciconcursos.com.br**
**26**
**BANCO DO BRASIL**
**AGENTE DE TECNOLOGIA - Microrregião 158 -TI GABARITO 1**
**70**
**Um método Java, chamado converte, deve receber uma string (str) como parâmetro e retornar uma string igual a str, exceto**
```

pelas letras minúsculas, que devem ser convertidas em letras maiúsculas.
Exemplo:
String recebida: "Abc $12d"
String retornada: "ABC $12D"
Qual método realiza essa tarefa?

(A) public static String converte(String str) {
String r= " ";
int i=0;

do {
char x=str.charAt(i);
if(Character.isLowerCase(x))
r=r.concat(Character.toString(Character.toUpperCase(x)));
else
r=r.concat(Character.toString(x));
i++;
} while(i < str.length());
return r;
}
(B) public static String converte(String str) {
String r=null;
int i=0;

while(i < str.length()) {
char x=str.charAt(i);
if(Character.isLowerCase(x))
r=r.concat(Character.toString(Character.toUpperCase(x)));
else
r=r.concat(Character.toString(x));
i++;
}
return r;
}
(C) public static String converte(String str) {
String r=null;
int i=0;

while(i < str.length()) {
if(str[i].isLowerCase(x))
r=r.concat(Character.toString(Character.toUpperCase(str[i])));
else
r=r.concat(Character.toString(str[i]));

```
i++;
}
return r;
}
```