Com certeza! Aqui está o texto com exatamente três quebras de linha entre a "Resposta Correta" e a "Explicação" para cada questão:

Simulação de Prova - Estruturas de Dados

Instruções: Leia atentamente cada questão e responda o que se pede.

Seção 1: Questões de Múltipla Escolha

(Instrução: Marque a alternativa que melhor responde à questão)

1. Analise o trecho de código Java do arquivo AULA 2 - Codigo que popula uma lista com n numeros aleatorios inteiros.java:

Java

```
if (!lista.contains(numero)) {
    lista.add(numero);
}
```

Qual a finalidade principal do método contains() neste contexto?
a) Ordenar os números na lista.
b) Verificar se um número já existe na lista antes de adicioná-lo, evitando duplicatas.
c) Remover um número específico da lista.
d) Contar quantos elementos existem na lista.

Resposta e Explicação:

Resposta Correta: b) Verificar se um número já existe na lista antes de adicioná-lo, evitando duplicatas.

Explicação: O método lista.contains(numero) retorna true se o número já estiver presente na lista, e false caso contrário. A condição if (!lista.contains(numero)) significa "se a lista NÃO contém o número". Portanto, o código dentro do if (lista.add(numero);) só será executado se o número ainda não existir na lista, garantindo assim que apenas números únicos sejam adicionados, prevenindo a inserção de duplicatas. As outras opções estão incorretas: sort() é usado para ordenar, remove() para remover, e size() para contar elementos.

2. No arquivo AULA 1 - INTRODUÇÃO ARQUIVO EM C .c, qual o propósito das variáveis vetorP e vetorI?

C

```
int vetor[10], *vetorP[10], *vetorI[10], i, tamP = 0, tamI = 0;
// ...
if(vetor[i] % 2 == 0) {
    vetorP[tamP++] = vetor[i]; // Linha relevante
} else {
    vetorI[tamI++] = vetor[i]; // Linha relevante
```

}
a) Armazenar todos os números do vetor original.
b) Armazenar ponteiros para os elementos pares (vetorP) e ímpares (vetorI) do vetor original. Nota: Há um erro conceitual no código original C, pois ele atribui inteiros a ponteiros. A questão avalia a intenção aparente.
c) vetorP armazena os índices pares e vetorI os índices ímpares.
d) São usados para calcular a média dos números pares e ímpares.

Resposta e Explicação:

Resposta Correta: b) Armazenar ponteiros para os elementos pares (vetorP) e ímpares (vetorI) do vetor original.

Explicação: As variáveis vetorP e vetorI são declaradas como arrays de ponteiros (*vetorP[10], *vetorI[10]). A lógica do código tenta separar os números do vetor original com base na paridade (par ou ímpar) e atribuí-los a posições nesses arrays de ponteiros. Embora a atribuição direta de um int (vetor[i]) a um int* (vetorP[tamP++]) esteja conceitualmente incorreta em C (deveria ser o endereço &vetor[i]), a intenção clara do código, dado o nome das variáveis e a lógica de separação, é armazenar referências (ou os próprios valores, devido ao erro) aos números pares em vetorP e aos ímpares em vetorI. As outras opções são incorretas: eles não armazenam todos os números, nem os índices, e não há cálculo de média nesse trecho.

3. Com base nas explicações em AULA 2 - Tarefas Da Aula.java, qual a principal diferença entre ArrayList e Vector em Java?
a) ArrayList é mais rápido para inserção e remoção, enquanto Vector é mais rápido para busca.
b) Vector não pode crescer dinamicamente, ao contrário do ArrayList.
c) Vector é sincronizado (thread-safe) e geralmente mais lento, enquanto ArrayList não é sincronizado e oferece melhor desempenho em ambientes single-thread.
d) ArrayList só armazena tipos primitivos, enquanto Vector armazena objetos.

Resposta e Explicação:

Resposta Correta: c) Vector é sincronizado (thread-safe) e geralmente mais lento, enquanto ArrayList não é sincronizado e oferece melhor desempenho em ambientes single-thread.

Explicação: A diferença fundamental mencionada historicamente e relevante em ambientes concorrentes é a sincronização. Métodos de Vector são synchronized, o que significa que apenas uma thread pode acessar o Vector por vez, tornando-o seguro para uso concorrente (thread-safe), mas introduzindo uma sobrecarga de desempenho. ArrayList não é sincronizado por padrão, o que o torna mais rápido em cenários onde apenas uma thread acessa a lista. Ambas as classes são baseadas em arrays, crescem dinamicamente e armazenam objetos (ou tipos primitivos via autoboxing). O desempenho de inserção/remoção/busca depende da posição, não sendo uma diferença fundamental fixa entre as duas como descrito na opção 'a'.

4. Qual estrutura de dados é utilizada no arquivo AULA 4 - DICIONARIOS.java para

associar nomes (chaves) a idades (valores)?

Java

```
Map<String, Integer> dicionario1 = new HashMap<>();
dicionario1.put("João", 25);
```

a) ArrayList
b) LinkedList
c) Vector
d) HashMap

Resposta e Explicação:

Resposta Correta: d) HashMap

Explicação: O código declara explicitamente Map<String, Integer> dicionario1 = new HashMap<>();. Map é uma interface em Java que representa uma coleção de pares chave-valor. HashMap é uma implementação concreta dessa interface que permite armazenar associações onde cada chave (neste caso, uma String representando o nome) mapeia para um valor (neste caso, um Integer representando a idade). O método put() é usado para inserir essas associações. ArrayList, LinkedList e Vector são implementações da interface List, que armazenam sequências ordenadas de elementos, não pares chave-valor.

5. No código da classe Aluno (presente nos arquivos da AULA 6 em Java, C# e Python), qual a importância de sobrescrever/implementar os métodos equals (Equals/__eq__) e hashCode (GetHashCode/__hash__)?
a) Para permitir a ordenação da lista de alunos pelo nome.
b) Para gerar o email do aluno automaticamente.
c) Para definir como comparar dois objetos Aluno (baseado no email, neste caso) e garantir o correto funcionamento em coleções como HashMap ou ao usar métodos como contains().
d) Para converter o objeto Aluno em uma representação String.

Resposta e Explicação:

Resposta Correta: c) Para definir como comparar dois objetos Aluno (baseado no email, neste caso) e garantir o correto funcionamento em coleções como HashMap ou ao usar métodos como contains().

Explicação: O método equals define o critério de igualdade entre dois objetos. No caso da classe Aluno nos arquivos da AULA 6, ele é implementado para comparar os emails, considerando dois alunos iguais se tiverem o mesmo email. O método hashCode deve ser consistente com equals: se dois objetos são iguais por equals, eles devem ter o mesmo hashCode. Essas implementações são cruciais para estruturas de dados baseadas em hash (como HashMap, HashSet) encontrarem objetos corretamente e para métodos como contains() em coleções saberem se um objeto específico já existe. A ordenação é feita pela interface Comparable ou por um Comparator (opção a). A geração de email é feita por outro método (opção b). A

conversão para String é feita pelo método toString (ToString/__str__) (opção d).

6. O código em AULA 4 - EXERCICIO CONTADOR DISCIPLINAS.java tem como objetivo principal:
a) Cadastrar alunos em diferentes disciplinas.
b) Calcular a média das notas dos alunos em cada disciplina.
c) Listar todos os alunos matriculados no curso de Ciência da Computação.
d) Contar em quantas disciplinas cada aluno está matriculado, percorrendo as listas de cada disciplina.

Resposta e Explicação:

Resposta Correta: d) Contar em quantas disciplinas cada aluno está matriculado, percorrendo as listas de cada disciplina.

Explicação: O código cria listas (ArrayList) para representar as matrículas em diferentes disciplinas (POO, ESTRUTURA, REDES). Ele então itera sobre uma lista de todos os alunos (listaGeral) e, para cada aluno, verifica (usando contains()) se ele está presente nas listas de cada disciplina. Um contador (cont) é incrementado para cada disciplina em que o aluno é encontrado. Por fim, imprime o nome do aluno e o número de disciplinas em que está matriculado (cont). As outras opções não correspondem à lógica principal do código: ele não cadastra alunos (eles já estão nas listas), não calcula médias, e não filtra especificamente por curso (embora use uma listaGeral).

Seção 2: Questões Discursivas

(Instrução: Responda de forma clara e concisa)

7. Explique a diferença fundamental entre ArrayList e LinkedList em Java, considerando a estrutura interna e o desempenho para operações comuns como adição/remoção no meio da lista e acesso a um elemento por índice. (Baseado em AULA 2 - Tarefas Da Aula.java)

Resposta e Explicação:

A diferença fundamental reside na estrutura de dados subjacente e como isso impacta o desempenho das operações:

ArrayList:

Estrutura Interna: Utiliza um array dinâmico internamente para armazenar os elementos. Os elementos são contíguos na memória (ou pelo menos o array que os contém é).
Acesso por Índice (get(index)): Muito Rápido (O(1) - tempo constante). Como é baseado em array, acessar um elemento pelo seu índice é uma operação direta de cálculo de endereço de memória.
Adição/Remoção no Meio: Lento (O(n) - tempo linear). Inserir ou remover um elemento no meio exige deslocar todos os elementos subsequentes para abrir espaço ou preencher a lacuna, respectivamente. Quanto maior a lista, mais lenta a operação.

Adição/Remoção no Final: Geralmente rápido (O(1) amortizado), mas pode ser O(n) se o array interno precisar ser redimensionado (copiar todos os elementos para um array maior).
LinkedList:

Estrutura Interna: Utiliza uma lista duplamente encadeada. Cada elemento (nó) armazena o próprio dado e referências (ponteiros) para o elemento anterior e o próximo na sequência. Os elementos não são necessariamente contíguos na memória.
Acesso por Índice (get(index)): Lento (O(n) - tempo linear). Para encontrar o elemento no índice i, a lista precisa ser percorrida sequencialmente a partir do início (ou fim, dependendo da implementação e do índice) até chegar à posição desejada.
Adição/Remoção no Meio (com Iterador ou referência ao nó): Muito Rápido (O(1) - tempo constante). Uma vez que se tem a referência ao nó onde a inserção/remoção ocorrerá, basta atualizar os ponteiros dos nós vizinhos, sem necessidade de deslocar outros elementos. A busca pelo nó ainda pode ser O(n) se não se tiver a referência direta.
Adição/Remoção no Início/Fim: Muito rápido (O(1)).
Uso de Memória: LinkedList geralmente consome um pouco mais de memória por elemento, pois precisa armazenar as referências extras para os nós anterior e próximo, além do dado em si.
Conclusão: Escolhe-se ArrayList quando o acesso frequente por índice é necessário e as inserções/remoções no meio são raras. Escolhe-se LinkedList quando inserções/remoções frequentes no meio (ou início/fim) são a principal operação, e o acesso por índice é menos comum.

8. Descreva o que o método gerarEmail (ou GerarEmail / gerar_email) na classe Aluno (AULA 6) faz. Analisando o código em Avaliaçao_0_EstruturaDeDados/Nomes.java, como o problema de emails potencialmente duplicados (mesmo primeiro e último nome) é tratado?

Resposta e Explicação:

O método gerarEmail (ou suas variantes em C#/Python) tem a função de criar um endereço de e-mail padronizado para um objeto Aluno. O processo geralmente envolve:

Obter o nome completo do aluno.
Dividir o nome completo em partes (normalmente usando o espaço como delimitador).
Pegar a primeira parte (primeiro nome) e a última parte (último nome).
Converter ambas as partes para letras minúsculas.
Concatenar o primeiro nome, um ponto (.), o último nome e o domínio fixo (@ufn.edu.br). Exemplo: Para "Ana Clara Souza", o resultado seria ana.souza@ufn.edu.br.
O código em Avaliaçao_0_EstruturaDeDados/Nomes.java demonstra uma estratégia para lidar com colisões de e-mail, que ocorrem quando dois alunos diferentes teriam o mesmo e-mail gerado por esse método padrão (por exemplo, dois alunos chamados "Ana Souza"). A abordagem utilizada é:

Geração Inicial: Gera o email base (ex: ana.souza@ufn.edu.br).
Verificação de Existência: Antes de atribuir o email ao aluno e adicioná-lo à coleção principal (nomesEmails, um HashMap que usa o email como chave), ele verifica se essa chave (o email gerado) já existe no HashMap usando nomesEmails.containsKey(email).
Tratamento de Duplicata (Loop): Se o email já existe, ele entra em um loop while.
Incremento Numérico: Dentro do loop, um contador numérico (j, começando em 2) é usado para criar variações do email. O email é reconstruído adicionando o número antes do "@". Ex: ana.souza2@ufn.edu.br.
Nova Verificação: A cada iteração, o contador j é incrementado (j++), e o novo email gerado (ex: ana.souza3@ufn.edu.br) é novamente verificado no HashMap.
Continuação do Loop: O loop continua até que seja gerado um email que não exista no HashMap.
Atribuição Única: Somente quando um email único é encontrado, ele é usado como chave para adicionar o aluno ao HashMap.
Essa estratégia garante que cada aluno no HashMap tenha um email (chave) único, mesmo que seus nomes resultassem inicialmente na mesma string de email.

Seção 3: Questões de Programação

(Instrução: Escreva o código solicitado na linguagem indicada)

9. (Java) Escreva um método estático em Java chamado filtrarPares que receba um ArrayList<Integer> como parâmetro e retorne um novo ArrayList<Integer> contendo apenas os números pares da lista original. A lista original não deve ser modificada.

Resposta e Explicação (Código Java):

Java

```
import java.util.ArrayList;

public class FiltroNumeros {

    /**
     * Filtra os números pares de uma lista de inteiros.
     *
     * @param original A lista de inteiros original (pode conter nulos).
     * @return Uma nova ArrayList contendo apenas os números pares da lista original.
     * Retorna uma lista vazia se a original for nula ou vazia.
     * Ignora elementos nulos na lista original.
     */
    public static ArrayList<Integer> filtrarPares(ArrayList<Integer> original) {
        // 1. Criar uma nova lista vazia para armazenar os pares.
        ArrayList<Integer> pares = new ArrayList<>();

        // 2. Verificar se a lista original não é nula para evitar NullPointerException.
        if (original == null) {
            return pares; // Retorna a lista vazia se a original for nula.
```

```
        }

        // 3. Iterar sobre cada elemento da lista original.
        for (Integer numero : original) {
            // 4. Verificar se o elemento atual não é nulo E se é par.
            // A verificação de nulidade é importante antes do operador %.
            if (numero != null && numero % 2 == 0) {
                // 5. Se for par, adicioná-lo à nova lista 'pares'.
                pares.add(numero);
            }
        }
        // 6. Retornar a nova lista contendo apenas os números pares.
        return pares;
    }

    public static void main(String[] args) {
        // Exemplo de uso
        ArrayList<Integer> numeros = new ArrayList<>();
        numeros.add(1);
        numeros.add(2);
        numeros.add(3);
        numeros.add(4);
        numeros.add(5);
        numeros.add(6);
        numeros.add(null); // Adiciona um nulo para teste
        numeros.add(8);
        numeros.add(-10); // Adiciona um par negativo

        System.out.println("Lista original: " + numeros);

        // Chama o método para filtrar os pares
        ArrayList<Integer> listaPares = filtrarPares(numeros);

        System.out.println("Lista de pares filtrada: " + listaPares); // Saída esperada: [2, 4,
6, 8, -10]

        // Teste com lista nula
        ArrayList<Integer> listaNula = null;
        System.out.println("Filtrando lista nula: " + filtrarPares(listaNula)); // Saída
esperada: []
    }
}
```

Explicação da Lógica: O método cria uma lista de resultados vazia. Ele percorre a lista de entrada elemento por elemento. Para cada elemento, verifica se não é nulo e se o resto da divisão por 2 é zero (indicando que é par). Se ambas as condições forem verdadeiras, o número é adicionado à lista de resultados. Finalmente, a lista de resultados contendo apenas os pares é retornada. A lista original permanece inalterada.

10. (Python ou C# - Escolha uma linguagem) Crie uma classe simples chamada Produto que possua os atributos codigo (inteiro) e nome (string). Em seguida, escreva um pequeno trecho de código (fora da classe, pode ser em uma função main ou similar) que:

Crie uma lista (List em C# ou list em Python) para armazenar objetos Produto.
Instancie (crie) 3 objetos Produto com dados diferentes.
Adicione esses 3 objetos à lista.
Percorra a lista e imprima os dados de cada produto (código e nome).
Resposta e Explicação (Código Python):

Python

```
# 1. Definição da classe Produto
class Produto:
    """Representa um produto com código e nome."""
    def __init__(self, codigo: int, nome: str):
        """Inicializador (construtor) da classe Produto."""
        self.codigo = codigo
        self.nome = nome

    def __str__(self) -> str:
        """Retorna uma representação em string do objeto Produto."""
        # Este método é chamado quando usamos print() no objeto.
        return f"Produto(Código: {self.codigo}, Nome: '{self.nome}')"

    def __repr__(self) -> str:
        """Retorna uma representação 'oficial' do objeto, útil para debugging."""
        # Muitas vezes é igual ou similar ao __str__ para classes simples.
        return f"Produto(codigo={self.codigo}, nome='{self.nome}')"

# 2. Código principal para criar e manipular a lista de produtos
if __name__ == "__main__":
    # Bloco principal, executado quando o script é rodado diretamente.

    # 3. Criar uma lista vazia para armazenar os produtos.
    lista_produtos: list[Produto] = []
    print(f"Lista inicial: {lista_produtos}")

    # 4. Instanciar (criar) 3 objetos Produto.
    produto1 = Produto(101, "Teclado Mecânico")
    produto2 = Produto(codigo=102, nome="Mouse Óptico") # Outra forma de passar
argumentos
    produto3 = Produto(205, "Monitor LED 27\"")

    # 5. Adicionar os objetos à lista.
    lista_produtos.append(produto1)
    lista_produtos.append(produto2)
    lista_produtos.append(produto3)
```

```
    print(f"Lista após adições: {lista_produtos}") # Usará o __repr__ dos objetos

    # 6. Percorrer a lista e imprimir os dados de cada produto.
    print("\n--- Detalhes dos Produtos na Lista ---")
    for i, item in enumerate(lista_produtos):
        # Usando o método __str__ definido na classe para impressão formatada:
        print(f"Item {i+1}: {item}")
        # Acessando os atributos diretamente (alternativa):
        # print(f"Item {i+1}: Código={item.codigo}, Nome='{item.nome}'")
    print("-------------------------------------")
```

Explicação da Lógica (Python): Primeiro, define-se a classe Produto com um construtor (__init__) que recebe codigo e nome e os armazena nos atributos do objeto (self.codigo, self.nome). O método __str__ é adicionado para fornecer uma representação legível do objeto ao ser impresso. No bloco principal (if __name__ == "__main__":), uma lista vazia é criada. Três instâncias da classe Produto são criadas com dados distintos. Cada instância (objeto) é adicionada à lista_produtos usando o método append(). Finalmente, um loop for percorre a lista_produtos, e para cada item (que é um objeto Produto), a função print() é chamada, que por sua vez invoca o método __str__ do objeto para exibir seus dados formatados.