Quais mudanças, quando feitas independentemente, garantem que o trecho de código a seguir imprima 100 durante a execução?

(Escolha todas as opções corretas.)

A. Alterar *data* para uma variável de instância e marcá-la como *volatile*.
B. Remover *parallel()* na operação de stream.
C. Alterar *forEach()* para *forEachOrdered()* na operação de stream.
D. Alterar *parallel()* para *serial()* na operação de stream.
E. Envolver o corpo da lambda com um bloco sincronizado.
F. O trecho de código sempre imprimirá 100 como está.

```
public class Exercicio_003 {

    public static void main(String[] args)
    {
        List<Integer> data = new ArrayList<>();
        IntStream.range(0,100).parallel().forEach(s -> data.add(s));
        System.out.println(data.size());
    }

}
```

Vamos analisar quais mudanças garantem que o trecho de código fornecido imprime 100 em tempo de execução.

```
List<Integer> data = new ArrayList<>();
IntStream.range(0, 100).parallel().forEach(s -> data.add(s));
System.out.println(data.size());
```

Análise das alterantivas

(A) Alterar data para uma variável de instância e marcá-la como volatile.
- Tornar data uma variável de instância e marcá-la como volatile não resolve os problemas de modificação concorrente. A palavra-chave volatile garante a visibilidade das alterações de variáveis entre threads, mas não evita problemas de concorrência.

(B) Remover parallel() na operação de stream.
- Remover parallel() faz com que a operação do stream seja sequencial. Assim, cada elemento será adicionado ao ArrayList um de cada vez, evitando problemas de concorrência.
- Mudança:
```
IntStream.range(0, 100).forEach(s -> data.add(s));
```

(C) Alterar forEach() para forEachOrdered() na operação de stream.
- forEachOrdered() garante a ordem de processamento em streams paralelos. No entanto, não resolve o problema de concorrência em si, mas pode ajudar a preservar a ordem em que os elementos são adicionados. Para garantir a impressão de 100, a lista ainda precisa ser thread-safe.
- Mudança:
```
IntStream.range(0, 100).parallel().forEachOrdered(s -> data.add(s));
```

(D) Alterar parallel() para serial() na operação de stream.
- Não existe o método serial() na API do Stream. O que podemos entender por essa opção é tornar o stream sequencial, o que é realizado removendo parallel(), como mencionado na opção B.

(E) Envolver o corpo da lambda com um bloco sincronizado.
- Envolver o corpo da lambda com um bloco synchronized garante que apenas um thread por vez possa adicionar elementos ao ArrayList, evitando problemas de concorrência.
- Mudança:
```
IntStream.range(0, 100).parallel().forEach(s -> {
    synchronized (data) {
        data.add(s);
    }
});
```

(F) O trecho de código sempre imprimirá 100 como está.
- Esta opção está incorreta. O trecho de código atual pode resultar em exceções de concorrência ou em uma contagem incorreta devido a modificações simultâneas no ArrayList.

RESPOSTA:

As mudanças que garantem que o trecho de código imprime 100 em tempo de execução são:

B. Remover parallel() na operação de stream.
C. Alterar forEach() para forEachOrdered() na operação de stream.
E. Envolver o corpo da lambda com um bloco sincronizado.