O que é verdade se o conteúdo do *path1* começar com o texto “Howdy”? (Escolha dois.)

```
System.out.println(Arquivos.mismatch(path1, path2));
```

A. Se path2 não existir, o código imprimirá -1.
B. Se path2 não existir, o código imprimirá 0.
C. Se path2 não existir, o código lança uma exceção.
D. Se o conteúdo do path2 começar com Hello, o código imprimirá -1.
E. Se o conteúdo de path2 começar com Hello, o código imprimirá 0.
F. Se o conteúdo de path2 começar com Hello, o código imprimirá 1.

BOYARSKY, Jeanne; SELIKOFF, Scott. **OCP Oracle® Certified Professional Java SE 17 Developer Study Guide**. John Wiley & Sons, 2022.

Vamos ver as opções possíveis para a questão fornecida. A questão envolve o uso do método `Files.mismatch(path1, path2)` da classe `Files` em Java, que é utilizado para comparar o conteúdo de dois arquivos (lembram-se daquela questão sobre `Arrays.mismatch()`? Mesmo comportamento). Esse método retorna o índice da primeira posição onde os dois arquivos diferem, ou `-1` se os arquivos forem idênticos. Agora, vamos examinar as opções:

Análise da Questão

A. Se path2 não existe, o código imprime -1.

- **Falso.** Se o arquivo `path2` não existe, o método `Files.mismatch` lançará uma exceção, pois não consegue abrir o arquivo para comparação.

B. Se path2 não existe, o código imprime 0.

- **Falso.** Novamente, como mencionado, a inexistência de `path2` resultará em uma exceção e não em um retorno de 0.

C. Se path2 não existe, o código lança uma exceção.

- **Verdadeiro.** Se `path2` não existe, uma exceção do tipo `java.nio.file.NoSuchFileException` será lançada.

D. Se o conteúdo de path2 começa com Hello, o código imprime -1.

- **Falso.** Para `Files.mismatch` retornar `-1`, os conteúdos dos dois arquivos devem ser idênticos. Como `path1` começa com "Howdy" e `path2` com "Hello", eles não são idênticos.

E. Se o conteúdo de path2 começa com Hello, o código imprime 0.

- **Falso.** O código imprimirá o índice da primeira diferença. Se `path1` começa com "Howdy" e `path2` começa com "Hello", a diferença é na primeira posição.

F. Se o conteúdo de path2 começa com Hello, o código imprime 1.

- **Verdadeiro.** A primeira diferença entre "Howdy" e "Hello" está na primeira posição (índice 0), onde "H" é igual em ambos, mas a segunda letra "o" (em "Howdy") difere da segunda letra "e" (em "Hello"). Então, a comparação difere na posição 1.

As respostas corretas são:

> **C. Se path2 não existe, o código lança uma exceção.**
> **F. Se o conteúdo de path2 começa com Hello, o código imprime 1.**

Vamos agora descrever a interface `Files`, seus principais métodos e para que ela serve, além de dar exemplos práticos de código para ilustrar melhor.

Interface `Files`

A classe `Files` pertence ao pacote `java.nio.file` e oferece métodos estáticos que lidam com operações comuns relacionadas a arquivos e diretórios, como criar, copiar, mover, excluir, ler, gravar e muito mais.

Principais Métodos da Classe `Files`

Aqui estão alguns dos métodos mais utilizados da classe `Files`:

- **`Files.exists(Path path)`**: Verifica se o arquivo ou diretório especificado existe.

```
Path path = Paths.get("example.txt");
boolean exists = Files.exists(path);
```

- **`Files.createFile(Path path)`**: Cria um novo arquivo.

```
Path path = Paths.get("newfile.txt");
Files.createFile(path);
```

- **`Files.createDirectory(Path path)`**: Cria um novo diretório.

```
Path path = Paths.get("newdir");
Files.createDirectory(path);
```

- **`Files.delete(Path path)`**: Exclui um arquivo ou diretório.

```
Path path = Paths.get("example.txt");
Files.delete(path);
```

- **`Files.copy(Path source, Path target, CopyOption... options)`**: Copia um arquivo ou diretório.

```
Path source = Paths.get("source.txt");
Path target = Paths.get("target.txt");
Files.copy(source, target, StandardCopyOption.REPLACE_EXISTING);
```

- **`Files.move(Path source, Path target, CopyOption... options)`**: Move ou renomeia um arquivo ou diretório.

```
Path source = Paths.get("source.txt");
Path target = Paths.get("target.txt");
Files.move(source, target, StandardCopyOption.REPLACE_EXISTING);
```

- **`Files.readAllBytes(Path path)`**: Lê todo o conteúdo de um arquivo e retorna um array de bytes.

```
Path path = Paths.get("example.txt");
byte[] bytes = Files.readAllBytes(path);
```

- **`Files.write(Path path, byte[] bytes, OpenOption... options)`**: Escreve bytes em um arquivo.

```
Path path = Paths.get("example.txt");
byte[] bytes = "Hello, world!".getBytes();
Files.write(path, bytes);
```

- **`Files.lines(Path path)`**: Lê todas as linhas de um arquivo e retorna um Stream de Strings.

```
Path path = Paths.get("example.txt");
try (Stream<String> lines = Files.lines(path)) {
    lines.forEach(System.out::println);
}
```

- **`Files.mismatch(Path path1, Path path2)`**: Compara o conteúdo de dois arquivos e retorna a posição da primeira diferença, ou `-1` se os arquivos forem idênticos.

```
Path path1 = Paths.get("file1.txt");
Path path2 = Paths.get("file2.txt");
long mismatchIndex = Files.mismatch(path1, path2);
System.out.println(mismatchIndex);
```

Métodos e Tratamento de Exceções

1. **`Files.exists(Path path)`**: Verifica se o arquivo ou diretório especificado existe.
    - **Retorno**: `true` se o arquivo ou diretório existir; `false` caso contrário.
    - **Exceções**: Pode lançar `SecurityException` se houver problemas de segurança ao acessar o arquivo.

2. **`Files.createFile(Path path)`**: Cria um novo arquivo.
    - **Retorno**: O caminho para o arquivo recém-criado.
    - **Exceções**:
        - Lança **FileAlreadyExistsException** se o arquivo já existir.
        - Lança **IOException** se houver problemas de E/S (entrada/saída) ou se o diretório pai não existir.
        - Lança **UnsupportedOperationException** se o array de atributos contiver um atributo que não pode ser configurado atomicamente ao criar o arquivo.
        - Lança **SecurityException** se um gerenciador de segurança estiver instalado e houver uma violação de segurança (por exemplo, falta de permissões de escrita).

3. **`Files.createDirectory(Path path)`**: Cria um novo diretório.
    - **Retorno**: O caminho para o diretório recém-criado.
    - **Exceções**:
        - Lança **FileAlreadyExistsException** se o diretório já existir.
        - Lança **IOException** se houver problemas de E/S (entrada/saída) ou se o diretório pai não existir.
        - Lança **UnsupportedOperationException** se o array de atributos contiver um atributo que não pode ser configurado atomicamente ao criar o diretório.
        - Lança **SecurityException** se um gerenciador de segurança estiver instalado e houver uma violação de segurança (por exemplo, falta de permissões de escrita).

4. **`Files.delete(Path path)`**: Exclui um arquivo ou diretório.
    - **Retorno**: Nada (void).
    - **Exceções**:
        - Lança **NoSuchFileException** se o arquivo ou diretório não existir.
        - Lança **DirectoryNotEmptyException** se o caminho for um diretório e ele não estiver vazio.
        - Lança **IOException** se houver problemas de E/S (entrada/saída) ao tentar deletar o arquivo ou diretório.

        - Lança **SecurityException** se um gerenciador de segurança estiver instalado e houver uma violação de segurança (por exemplo, falta de permissões de deleção).

5. **`Files.copy(Path source, Path target, CopyOption... options)`**: Copia um arquivo ou diretório.
    - **Retorno**: O caminho para o arquivo de destino.
    - **Exceções**:
        - Lança **IOException** se o arquivo de origem não existir.
        - Lança **FileAlreadyExistsException** se o arquivo de destino já existir (a menos que seja especificada a opção `StandardCopyOption.REPLACE_EXISTING`).
        - Lança **DirectoryNotEmptyException** se o diretório de destino não estiver vazio (a menos que seja especificada a opção `StandardCopyOption.REPLACE_EXISTING`).
        - Lança **SecurityException** se uma operação de segurança for violada.
        - Lança **UnsupportedOperationException** se a opção especificada não for suportada.
        - Lança **IOException** se houver outros problemas de E/S.

6. **`Files.move(Path source, Path target, CopyOption... options)`**: Move ou renomeia um arquivo ou diretório.
    - **Retorno**: O caminho para o arquivo de destino.
    - **Exceções**:
        - Lança **IOException** se o arquivo de origem não existir.
        - Lança **FileAlreadyExistsException** se o arquivo de destino já existir (a menos que `StandardCopyOption.REPLACE_EXISTING` seja especificado).
        - Lança **DirectoryNotEmptyException** se o diretório de destino não estiver vazio (a menos que `StandardCopyOption.REPLACE_EXISTING` seja especificado).
        - Lança **AtomicMoveNotSupportedException** se a operação de movimento não for suportada atomicamente pelo sistema de arquivos.
        - Lança **SecurityException** se uma operação de segurança for violada.
        - Lança **UnsupportedOperationException** se a opção especificada não for suportada.
        - Lança **IOException** se houver outros problemas de E/S.

7. **`Files.readAllBytes(Path path)`**: Lê todo o conteúdo de um arquivo e retorna um array de bytes.
    - **Retorno**: Um array de bytes contendo o conteúdo do arquivo.
    - **Exceções**:
        - Lança **IOException** se ocorrer um erro de E/S ao ler o arquivo.
        - Lança **OutOfMemoryError** se o tamanho do arquivo for maior que o espaço disponível na memória.
        - Lança **SecurityException** se uma operação de segurança for violada.

8. **`Files.write(Path path, byte[] bytes, OpenOption... options)`**: Escreve bytes em um arquivo.
    - **Retorno**: O caminho para o arquivo.
    - **Exceções**:
        - Lança **IOException** se ocorrer um erro de E/S ao escrever ou criar o arquivo, ou se o texto não puder ser codificado usando o conjunto de caracteres especificado.
        - Lança **IllegalArgumentException** se as opções contiverem uma combinação inválida de opções.
        - Lança **UnsupportedOperationException** se uma opção não suportada for especificada.
        - Lança **FileAlreadyExistsException** se um arquivo com esse nome já existir e a opção `StandardOpenOption.CREATE_NEW` for especificada (exceção específica opcional).
        - Lança **SecurityException** se, no caso do provedor padrão e se um gerenciador de segurança estiver instalado, o método *checkWrite* for invocado para verificar o acesso de escrita ao arquivo. O método *checkDelete* é invocado para verificar o acesso de exclusão se o arquivo for aberto com a opção `StandardOpenOption.DELETE_ON_CLOSE`.

9. **`Files.lines(Path path)`**: Lê todas as linhas de um arquivo e retorna um `Stream` de Strings.
    - **Retorno**: Um `Stream` de Strings, onde cada elemento é uma linha do arquivo.
    - **Exceções**:
        - Lança **IOException** se ocorrer um erro de E/S ao abrir o arquivo.
        - Lança **SecurityException** se, no caso do provedor padrão e se um gerenciador de segurança estiver instalado, o método checkRead for invocado para verificar o acesso de leitura ao arquivo.

10. **`Files.mismatch(Path path1, Path path2)`**: Compara o conteúdo de dois arquivos e retorna a posição da primeira diferença, ou `-1` se os arquivos forem idênticos.
    - **Retorno**: O índice da primeira posição onde os arquivos diferem, ou `-1` se forem idênticos.
    - **Exceções**:
        - Lança **IOException** se ocorrer um erro de E/S.

- Lança **SecurityException** se, no caso do provedor padrão e se um gerenciador de segurança estiver instalado, o método checkRead for invocado para verificar o acesso de leitura aos dois arquivos.

Um exemplo simples de uso dos métodos da classe **`Files`**:

```
public static void main(String[] args) {
    Path path1 = Paths.get("file1.txt");
    Path path2 = Paths.get("file2.txt");

    try {
        // Cria os arquivos se não existirem
        if (!Files.exists(path1)) {
            Files.createFile(path1);
            Files.write(path1, "Howdy!".getBytes());
        }
        if (!Files.exists(path2)) {
            Files.createFile(path2);
            Files.write(path2, "Hello!".getBytes());
        }

        // Lê e imprime o conteúdo dos arquivos
        System.out.println("Conteúdo de file1.txt:");
        Files.lines(path1).forEach(System.out::println);

        System.out.println("Conteúdo de file2.txt:");
        Files.lines(path2).forEach(System.out::println);

        // Compara os arquivos
        long mismatchIndex = Files.mismatch(path1, path2);
        System.out.println("Primeira diferença na posição: " + mismat-
chIndex);

    } catch (NoSuchFileException e) {
        System.err.println("Arquivo não encontrado: " + e.getFile());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Este código Java realiza as seguintes operações:

1. Define dois caminhos (`Path`) para dois arquivos de texto: `file1.txt` e `file2.txt`.
2. Verifica se os arquivos existem. Se não existirem, cria-os e escreve conteúdos neles.
3. Lê e imprime o conteúdo de cada arquivo.
4. Compara os conteúdos dos arquivos `file1.txt` e `file2.txt` e identifica a primeira posição onde há uma diferença.

Neste código duas *exceptions* são tratadas:

- `NoSuchFileException`: Captura o caso em que um dos arquivos (`file1.txt` ou `file2.txt`) não existe.
- `IOException`: Trata outras exceções de entrada e saída que podem ocorrer durante a criação, escrita, leitura ou comparação dos arquivos.

No console teremos:

```
Conteúdo de file1.txt:

Howdy!

Conteúdo de file2.txt:

Hello!

Primeira diferença na posição: 1
```