1. Unterschied zwischen `==` und `equals()`

- `==` **Operator**:
  - Vergleicht die Speicheradressen (Referenzen) der Objekte
  - Prüft, ob beide Variablen auf dasselbe Objekt im Memory zeigen
  - Nicht geeignet für inhaltlichen String-Vergleich
- `equals()` **Methode**:
  - Vergleicht den tatsächlichen Inhalt der Strings
  - Prüft Zeichen für Zeichen ob die Strings identisch sind
  - Korrekte Methode für String-Inhaltsvergleich

2. Beispiel warum `==` unzuverlässig ist

java Copy Download

```java
String a = "hello";
String b = new String("hello");
String c = "hello";

System.out.println(a == b);  // false (unterschiedliche Objekte)
System.out.println(a == c);  // true (selbes Objekt im String-Pool)
System.out.println(a.equals(b)); // true (gleicher Inhalt)
```

3. Warum das gezeigte Beispiel `true` zurückgibt

java Copy Download

```java
String string1 = "abc";
String string2 = "abc";
System.out.println(string1 == string2); // true
```

Grund: Beide Strings werden aus dem **String-Pool** genommen. Java optimiert String-Literale und verwendet dasselbe Objekt für identische Literale.

4. Mit `new String("abc")`

java Copy Download

```java
String string1 = "abc";
String string2 = new String("abc");
System.out.println(string1 == string2); // false
```

Grund: `new` erzwingt die Erstellung eines neuen String-Objekts im Heap, außerhalb des String-Pools.

Visualisierung des Memory:

Copy Download

```
String-Pool: ["abc"] <-- string1, string2 (im ersten Beispiel)
Heap:        [new String("abc")] <-- string2 (im zweiten Beispiel)
```

Best Practices:

1. **Immer** `equals()` für String-Vergleiche verwenden:

   java Copy Download

   ```java
   if (string1.equals(string2)) { ... }
   ```

2. Für case-insensitive Vergleiche:

   java Copy Download

   ```java
   if (string1.equalsIgnoreCase(string2)) { ... }
   ```

3. **String-Pool Nutzung** mit `intern()`:

   java Copy Download

   ```java
   String s = new String("abc").intern(); // holt aus Pool oder fügt hinzu
   ```

Warum gibt es diesen Unterschied?

- **Performance-Optimierung**: Der String-Pool vermeidet Duplikate von Literalen
- **Objektidentität vs. Wertgleichheit**: `==` prüft Objektidentität, `equals()` Wertgleichheit

Diese Unterscheidung ist fundamental in Java und gilt für alle Objekttypen (nicht nur Strings), während `==` bei primitiven Datentypen (int, char etc.) den Wert vergleicht.