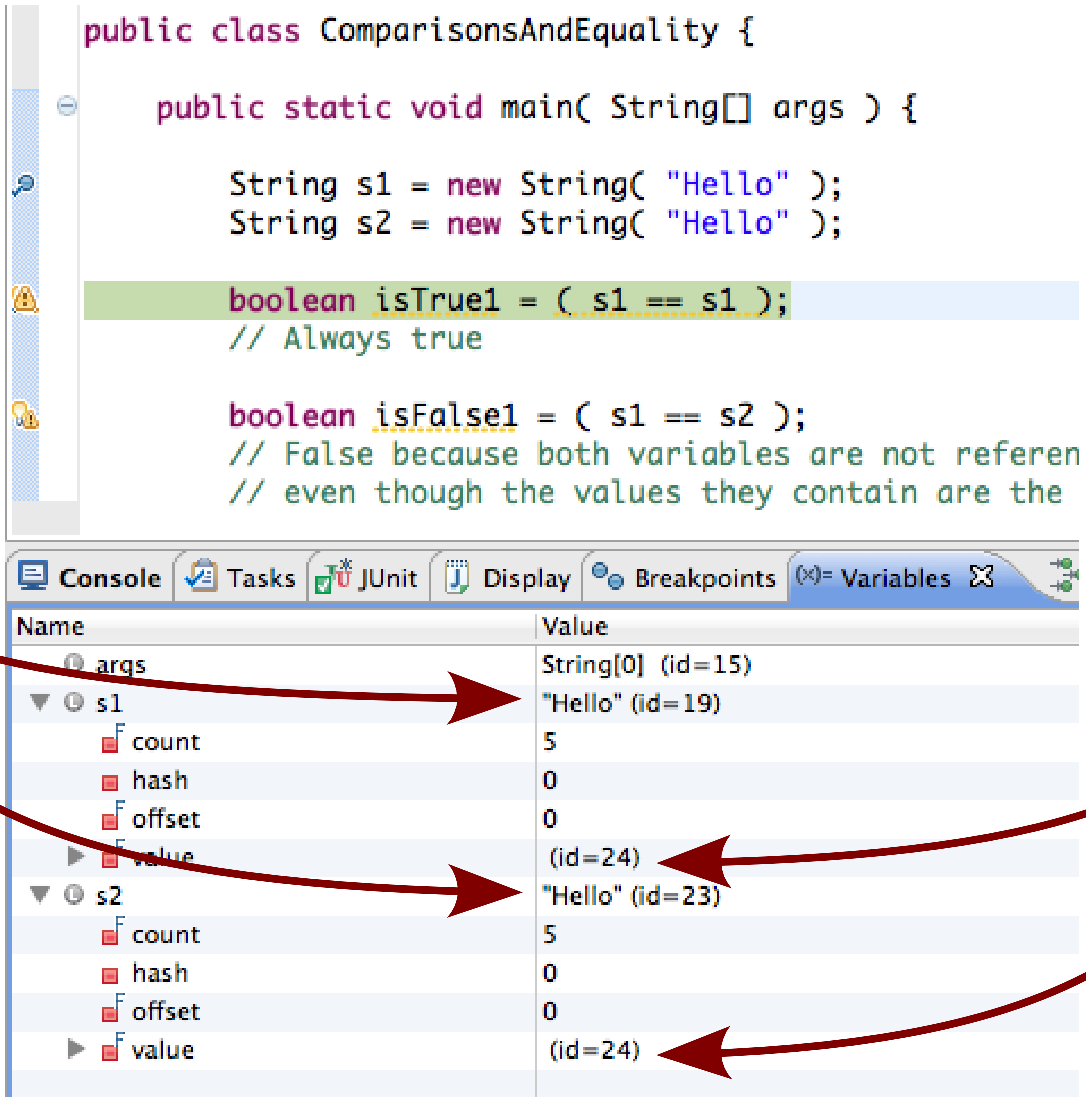
Question from first lecture

In the example called ComparisonAndEquality, the first assignment of String variables s1 and s2 results in the allocation of two different memory locations for the variables and two identical memory locations for their internal state, an optimization.

However, in the second assignment of String variables s1 and s2, both the memory locations for the variables and their internal states are the same, an even more significant optimization.

If Java could do that in the second case, why didn't it do it in the first case?

String allocation with 'new'



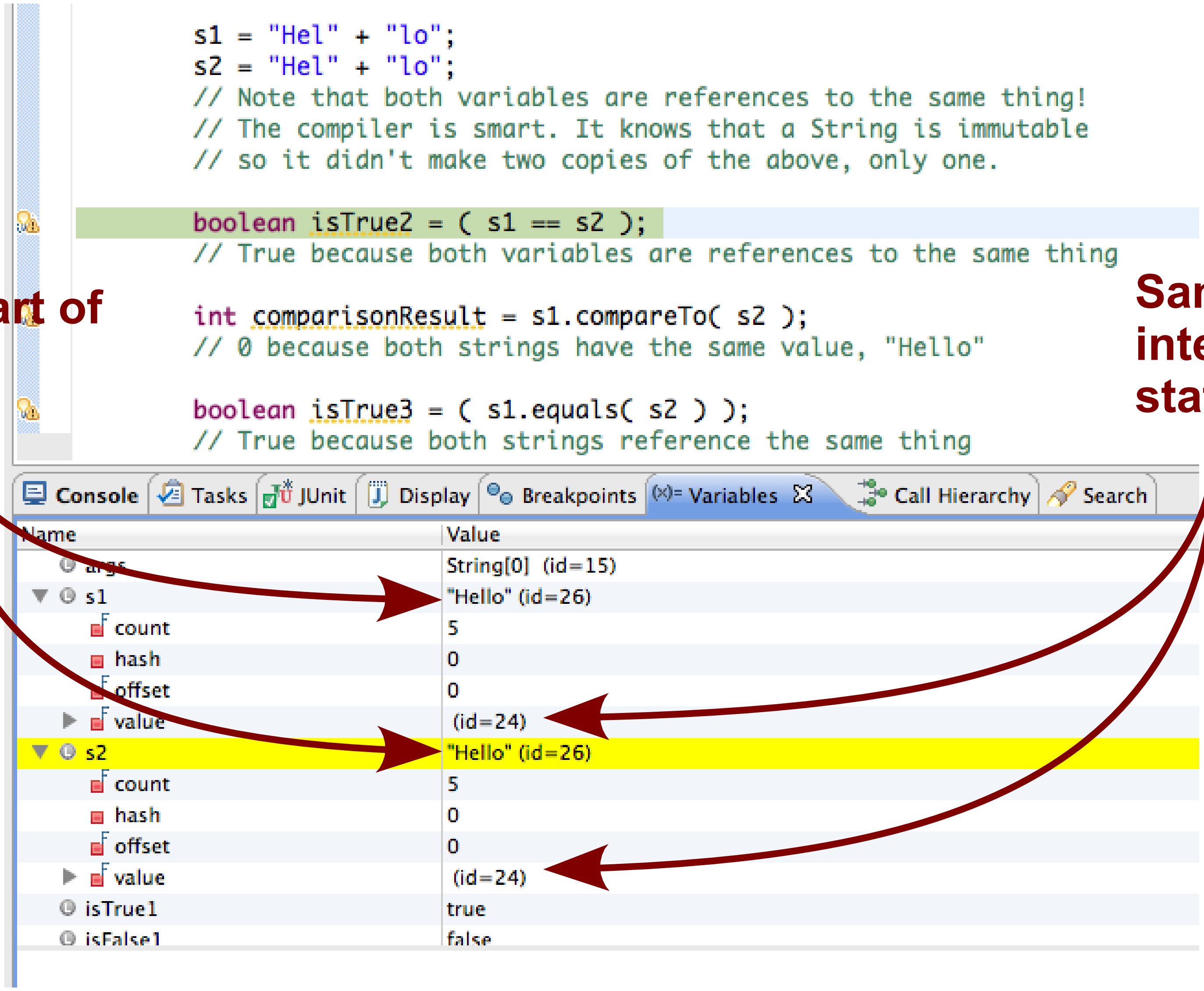
```
public class ComparisonsAndEquality {

    public static void main( String[] args ) {

        String s1 = new String( "Hello" );
        String s2 = new String( "Hello" );

        boolean isTrue1 = ( s1 == s1 );
        // Always true

        boolean isFalse1 = ( s1 == s2 );
        // False because both variables are not referen
        // even though the values they contain are the
```

Name	Value
args	String[0] (id=15)
s1	"Hello" (id=19)
count	5
hash	0
offset	0
value	(id=24)
s2	"Hello" (id=23)
count	5
hash	0
offset	0
value	(id=24)

Different parts of memory

Same internal state

Strings assigned to constants



```
s1 = "Hel" + "lo";
s2 = "Hel" + "lo";
// Note that both variables are references to the same thing!
// The compiler is smart. It knows that a String is immutable
// so it didn't make two copies of the above, only one.

boolean isTrue2 = ( s1 == s2 );
// True because both variables are references to the same thing

int comparisonResult = s1.compareTo( s2 );
// 0 because both strings have the same value, "Hello"

boolean isTrue3 = ( s1.equals( s2 ) );
// True because both strings reference the same thing
```

Name	Value
args	String[0] (id=15)
s1	"Hello" (id=26)
count	5
hash	0
offset	0
value	(id=24)
s2	"Hello" (id=26)
count	5
hash	0
offset	0
value	(id=24)
isTrue1	true
isFalse1	false

Same part of memory

Same internal state

Imagine the following method

```
public SomeData getData( String name ) {
    for( SomeData data : _dataList ) {
        If( data.getName() == name )
            return data;
        }
        return null;
    }
}
```

We are using the identity of the string to find data in a list. We don't care whether the values are the same. There could be many "John Smith" entries.