HW

Date: ___/___/___

01 [1] Comparison between a.equals(b); &

a==b →

a==b	a.equals(b);
(i) This checks if both a and b point to the same object in memory	(i) This checks if the values inside a and b are equal
(ii) Even if a and b have the same value inside, if they are two different objects this will return false.	(ii) Works only if the class has overridden the equals() method to compare content. Most built-in Java classes.
(iii) a==b → checks if both are exactly the same object.	(iii) a.equals(b); → checks if the contents are the same.

(iv) Example:
string a = new string("hello");
string b = new string("Hello");
system.out.println(a==b);
OP: false (not same object)
	(iv) Example:
	string a = new string("hello");
	string b = new string("hello");
	system.out.println(a.equals(b));
	OP
	true (same content)

02 Java strings are immutable but why?

⇒ Here,

(i). A new string "hello world" is created.

(ii) The variable s now points to the new string.

(iii) The old "hello" still exists in memory.

The original strings immutable in Java :-

(i) String pool / memory efficiency: Java uses a string pool to save memory.

Example:

```
string a = "Java";
string b = "Java";          [true]
system.out.println(a==b);
```

(ii) Simplicity and safety: strings are used every where in Java. Keeping them immutable avoids many bugs and confusing behavior.

(iii) Caching: Immutable objects can be safely cached or stored in collections like HashMap.

Example:

```
Map<String, String> map = new HashMap<
();
map.put("name", "Rahim");
```

If "name" changed often storing, we could lose our data.

(iv) Thread safety: Since strings can't change, multiple threads can safely use the same string at the same ~~the~~ time - no risk of unexpected changes.

(v) Security reasons: Java uses strings in many secure please:-

- Usernames. Passwords. File paths. Class names.

If a string could be changed, a hacker might

change the behavior of code.

Example:

```
class.forName("com.bank.Account");
```

———— X ————