The "==" operator is meant for comparing values, which can be:

- Either those stored in a primitive variable (i.e., int, long, float, double, char, and boolean);
- Or those stored in a reference variable (i.e., the address of some object stored in the memory)

In the case of comparing primitive values:

```
===============
int i = 5;
int j = i;
System.out.println(i == j); /* which will return true */
===============
```

Line 2 above copies what's stored in i into j. Consequently, variables i and j store DIFFERENT copies of the same value of 5. If you reassign one, the other will not be affected. For example:

```
===============
/* continued from the above code involving i and j */
i = 7; /* reassign i, which does not affect j */
System.out.println(i == j); /* which will return false */
===============
```

And, that's the end of story of comparing primitive values using "==".

In the case of comparing addresses stored in reference variables:

```
===============
Person p1 = new Person("Jim", 40);
Person p2 = p1;
System.out.println(p1 == p2); /* which will return true */
===============
```

Line 2 above copies what's stored in p1 into p2. Consequently, variables p1 and p2 store DIFFERENT copies of the same person object's address. When two reference variables (e.g., p1 and p2) store copies of the same address, conceptually they point to the same object, and this is called aliasing. When you try to change the attribute value of the object being pointed to, you can use either p1 or p2 as the context object, and the change effect is visible to both variables. For example:

```
===============
/* continued from the above code involving p1 and p2 */
System.out.println(p1.age); /* 40 */
System.out.println(p2.age); /* also 40, because of aliasing */
p1.setAge(50); /* change the age of person object pointed to by p1 (and p2) */
System.out.println(p1.age); /* 50 */
System.out.println(p2.age); /* also 50, because of aliasing */
===============
```

Nonetheless, similar to the case of primitive variables, if you reassign a reference variable, by storing a different address, the other will not be affected. For example:

```
==============
/* continued from the above code involving p1 and p2 */
p2 = new Person("Jonathan", 60); /* reassign p2, which does not affect p1 */
System.out.println(p1 == p2); /* which will return false */
p1.setAge(70); /* change the age of person object pointed to by p1 (but not by p2 anymore) */
System.out.println(p1.age); /* 70 */
System.out.println(p2.age); /* still 60, because there's no aliasing */
==============
```

And, that's the end of story of comparing primitive values using "==".

So you can see that "==" only allows us to compare the stored addresses of two reference variables. But it's often the case that we want to compare two objects not by their addresses, but instead their "contents". For example:

```
==============
Person p3 = new Person("Alan", 65);
Person p4 = new Person("Alan", 65);
System.out.println(p3 == p4); /* false, because p3 and p4 point to separately-created objects
*/
==============
```

In the above case, we may want to have a way to checking whether two person objects, with distinct addresses, are equal based on their names and ages.

The "equals" METHOD (rather than operator) is how Java handles object equality, which can be customized by you as a programmer. Syntactically, only REFERENCE variables (of a class type such as String, Person, Game, etc.) can be applied the "equals" method, whereas it's illegal to use the "equals" method on primitive variables. For example:

```
==============
int m = 8;
int n = 8;
boolean mEqualN = m.equals(n); /* not compiling, because m cannot be the context object */
==============
```

For now, the only objects you need to apply the "equals" method to compare "contents" are strings. For example:

```
==============
Scanner input = new Scanner(System.in);
System.out.println("Enter 1st string:");
String s1 = input.nextLine();
System.out.println("Enter 2nd string:");
String s2 = input.nextLine();
System.out.println("s1 and s2 point to same object: " + (s1 == s2));
System.out.println("s1 and s2 have same contents: " + s1.equals(s2));
==============
```

Executing the above program, which prompts you for two string values, the first print statement always outputs FALSE, because two separate string objects are pointed by s1 and s2, respectively, whereas the second print statement outputs whether or not the two string objects (pointed to by s1 and s2) have the same sequence of characters. Say you enter "York" followed by "York", then the output should be:

```
==============
false
true
==============
```

On the other hand, if you enter "York" followed by "University", then the output should be:

```
==============
false
false
==============
```