Reference vs. Value Comparison

When we compare value types with `==`, the C# compiler performs a *value* comparison. For example, this prints `true` because the value `6` is equal to the value `6`:

```
int int1 = 6;
int int2 = 6;
Console.WriteLine(int1 == int2);
// Output: true
```

`int1` and `int2` are the actual value `6`. When we compare the value `6` with `6`, they're the same!

When we compare reference types with `==`, the C# compiler performs a *referential*comparison, which means it checks if two variables refer to the same memory location. For example, this prints `false` because `d1` and `d2` refer to two different locations in memory (even though they contain objects with the same values):

```
Dissertation d1 = new Dissertation(50);
Dissertation d2 = new Dissertation(50);
Console.WriteLine(d1 == d2);
// Output: false
```

We constructed two different `Dissertation` objects which happened to have the same values. Each reference (`d1` and `d2`) point to different objects, so they are not equal.

☑**Instructions**

1.
Create a variable `b1` that refers to a new `Book` object.

Hint

Here's an example of constructing a Random object:

```
Random rand = new Random();
```

2.
Create a variable `b2` that holds the same reference as `b1`.

Hint

Remember that any variable representing an object is a reference to the object, not the object itself.

3.
Print the result of `b1 == b2` to the console. Why is that the value?

Hint

The result is `true` because both variables refer to the same location in memory.

(`True` is printed even though `true` is the value. It's just the compiler formatting for the console.)