Review

You made it! References aren't always easy, but learning how to use them unlocks a whole new set of superpowers in C#.

In this lesson you learned that:

- Classes and interfaces are *reference types*. A variable of this type holds a reference to the data, not the data itself. This is different from *value types* like `int` and `bool`
- The equality operator (`==`) uses a *referential*comparison for reference types and a *value* comparison for value types
- Multiple references can be created for a single object
- A reference and its corresponding object do not have to be the same type. For example, we can refer to a subclass object by an inherited superclass or implemented interface reference
- The functionality available to an object reference is determined by the reference's type, not the object's type
- *Polymorphism* is the ability in programming to present the same interface for differing data types
- Referencing an object by an inherited type or implemented interface is called *upcasting*. It can be done implicitly
- Referencing an object by a derived class is called *downcasting*, which must be made explicit by adding the type name in parentheses. It may cause an `InvalidCastException` error when the code is run
- To signify that a reference is "empty" or refers to no object, we set it equal to `null`
- If a reference is not set to any value it is *unassigned* and cannot perform any operations

☑**Instructions**

In **Program.cs**, there are two lines that are commented out:

```
f.Define();
```

```
bdiss3.Define();
```

Before you move on, make sure you can explain why each of them cause an error.