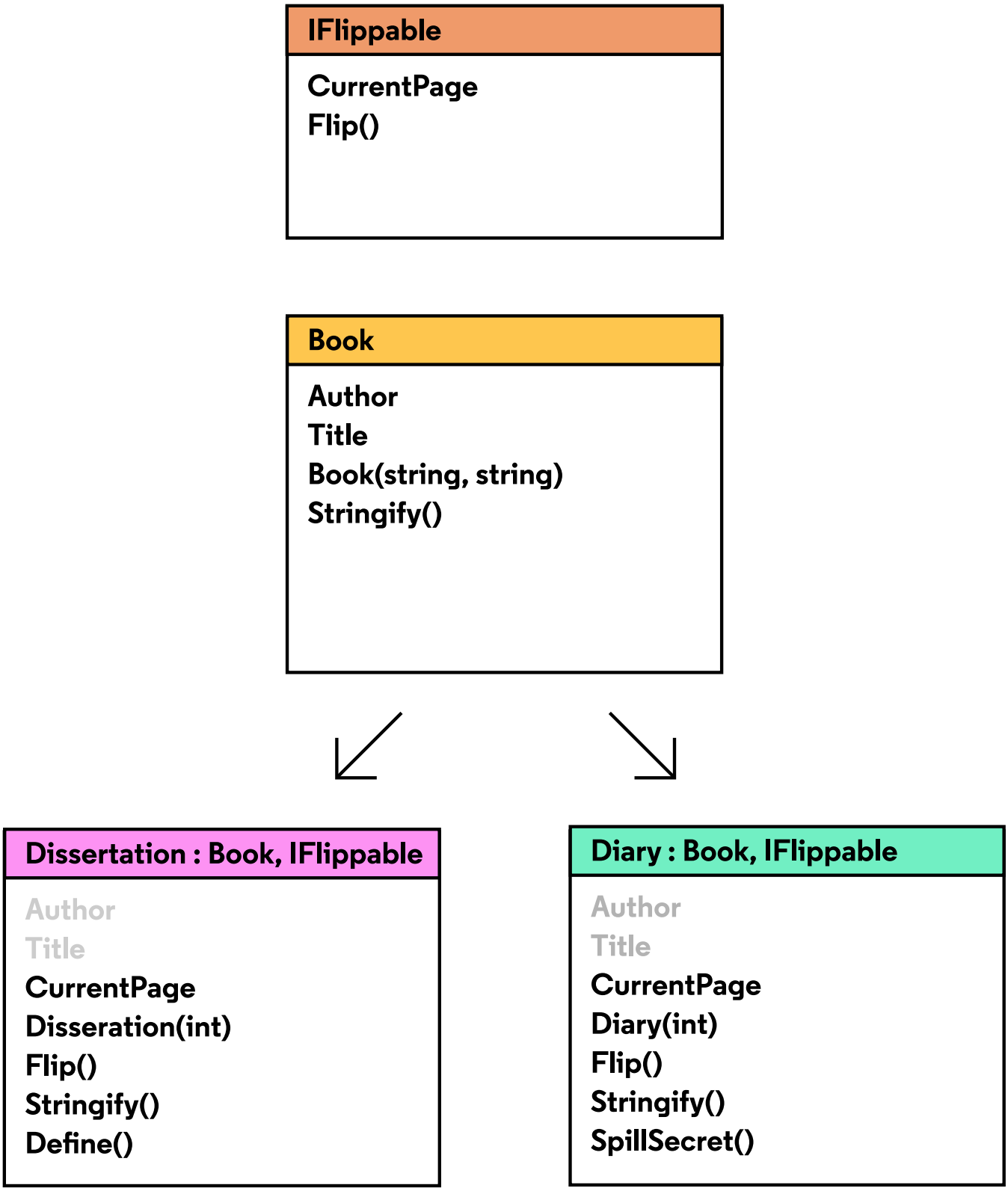
Casting

So far we've referred to objects with a reference of their own type, an inherited type, or an implemented interface:

```
Dissertation diss = new Dissertation();
Book bdiss = diss;
IFlippable fdiss = diss;
```

The process is called *upcasting*. As we saw in the last exercise upcasting allows us to work with multiple types at once. It also lets us safely store an object without knowing its specific type. You can think of upcasting as using a reference "up" the inheritance hierarchy:



What happens if you try to *downcast*, or reference an object by a subclass? You'll need to do this when you want to access the specific functionality of a subclass.

For example what happens when we refer to a `Book` object as a `Dissertation` type?

```
Book bk = new Book();
Dissertation dbk = bk;
// Error!
```

The code produces this error:

```
error CS0266: Cannot implicitly convert type `Book` to `Dissertation`. An
explicit conversion exists (are you missing a cast?)
```

Not every downcast is possible in C#. In this case, `Dissertation` has a `Define()` method that is incompatible with `Book`. This is the computer's way of telling you: there's a chance that this cast won't work!

To get around this error, we must explicitly downcast, like below. The desired type is written in parentheses:

```
Book bk = new Book();
Dissertation bdk = (Book)bk;
```

This essentially tells the computer: "I know the risk I'm taking, and this might fail if I'm not careful."

In many cases, the downcast will still fail. Here, `Book` can't reference a `Dissertation` object, so when we run explicitly downcast we see that it fails with a new error message:

```
System.InvalidCastException: Specified cast is not valid.
```

There are multiple ways to deal with downcasting, including the `as` and `is` operators. We won't get into those now, but you can learn about them in the Microsoft C# Programming Guide: Casting and type conversions if you'd like. For now, focus on these things:

> *Upcasting* is creating a superclass or interface reference from a subclass reference
>
> *Downcasting* is creating a subclass reference from a superclass or interface reference.

Upcasting can be done implicitly, while downcasting cannot

☑Instructions

1.

In **Program.cs**, `Dissertation` and `Diary` objects are being EXPLICITLY upcast to the `Book` type. We know that those explicit casts aren't necessary.

Delete the explicit casts from both lines.

Hint

Here's an example of explicit casting and implicit casting:

```
// Explicit
Truck pickup = new Truck();
Vehicle vpickup = (Vehicle)pickup;
```

```
// Implicit
Truck pickup = new Truck();
Vehicle vpickup = pickup;
```