Arrays of References

We know that we can use inherited classes and implemented interfaces to reference an object:

```
Dissertation diss = new Dissertation(50);
IFlippable fdiss = diss;
```

This allows to work with many similar types at the same time. Imagine if we didn't have this feature and we had to "flip" a group of `Diary` and `Dissertation` types:

```
Diary dy1 = new Diary(1);
Diary dy2 = new Diary(30);
Dissertation diss1 = new Dissertation(50);
Dissertation diss2 = new Dissertation(49);
dy1.Flip();
dy2.Flip();
diss1.Flip();
diss2.Flip();
```

Look at all that code! It would be faster and safer if we could store the references in an array and loop through it. But would it be an array of `Diary[]` or an array of `Dissertation[]` or something else? Since both dissertations and diaries are flippable (they both implement the `IFlippable` interface), we can create references to them as `IFlippable`s:

```
IFlippable f1 = new Diary(1);
IFlippable f2 = new Diary(30);
IFlippable f3 = new Dissertation(50);
IFlippable f4 = new Dissertation(49);
```

Instead of dealing with individual variables, we can use an array of `IFlippable` references:

```
IFlippable[] classroom = new IFlippable { new Diary(1), new Diary(30), new
Dissertation(50), new Dissertation(49) };
```

Then to “flip” each element, we can write a `foreach` loop:

```
foreach (IFlippable f in classroom)
{
  f.Flip();
}
```

We can only access the functionality defined in the interface. For example, we couldn’t access `f.Title` because `Title` isn’t a property defined in `IFlippable`.

☑Instructions

1.

Create a variable `books` of type `Book[]` that contains `diss1`, `diss2`, `dy1`, and `dy2`.

Hint

There are (at least) two ways of declaring an array. In this example we make an array of type `int[]`:

```
int[] plantHeights = new int[] { 3, 4, 6 };
```

or

```
int[] plantHeights = { 3, 4, 6 };
```

2.

Make an empty `foreach` loop that loops through each element in the array.

Hint

Here's an example `foreach` loop that loops through each `Forest` in the `parks` array:

```
foreach (Forest f in parks)
{
}
```

3.
In the body of the loop, print out the `Title` of each element.

Hint

Here's an example of accessing the `Age` property for each `Forest` in `parks`:

```
foreach (Forest f in parks)
{
  Console.WriteLine(f.Age);
}
```