Properties

As of now, a program can plant any value in a `Forest` field. For example, if we had an `area` field of type `int`, we could set it to `0`, `40`, or `-1249`. Can we have a forest of -1249 area? We need a way to define what values are valid and disallow those that are not. C# provides a tool for that: *properties*.

Properties are another type of class member. Each property is like a spokesperson for a field: it controls the access (getting and setting) to that field. We can use this to validate values before they are set to a field. A property is made up of two methods:

- a `get()` method, or getter: called when the property is accessed
- a `set()` method, or setter: called when the property is assigned a value

This shows a basic `Area` property without validation:

```
public int area;
public int Area
{
  get { return area; }
  set { area = value; }
}
```

The `Area` property is associated with the `area` field. It's common to name a property with the title-cased version of its field's name, e.g. `age` and `Age`, `name` and `Name`.

The `set()` method above uses the keyword `value`, which represents the value we assign to the property. Back in **Program.cs**, when we access the `Age` property, the `get()` and `set()` methods are called:

```
Forest f = new Forest();
f.Area = -1; // set() is called
Console.WriteLine(f.Area); // get() is called; prints -1
```

In the above example, when `set()` is called, the `value` variable is -1, so `area` is set to -1.

Here's the same property with validation in the `set()` method. If we try to set `Area` to a negative value, it will be changed to `0`.

```
public int Area
{
  get { return area; }
  set
  {
    if (value < 0) { area = 0; }
    else { area = value; }
  }
}
```

In **Program.cs**:

```
Forest f = new Forest();
// set() is called
f.Area = -1;
// get() is called; prints 0
Console.WriteLine(f.Area);
```

☑**Instructions**

1.

Define a basic `Name` property for the `name` field. The getter and setter will have no validation.

Hint

To define an `Area` property:

```
public int Area
{
  get { return area; }
  set { area = value; }
}
```

2.

Define a basic `Trees` property for the `trees` field. The getter and setter will have no validation.

Hint

To define an `Area` property:

```
public int Area
{
  get { return area; }
  set { area = value; }
}
```

3.
In **Program.cs**,

Replace any use of the field `f.name` with the property `f.Name`.

Replace any use of the field `f.trees` with the property `f.Trees`.

4.
Define a `Biome` property for the `biome` field. It will have a getter and setter. The setter should only allow three values: `"Tropical"`, `"Temperate"`, and `"Boreal"`. If any other value is used, set `biome` to `"Unknown"`.

For example, this is how it would work in **Program.cs**:

```
f.Biome = "Tropical";
// Prints Tropical
Console.WriteLine(f.Biome);

f.Biome = "Desert";
// Prints Unknown
Console.WriteLine(f.Biome);
```

Hint

Here's an example of an `Area` property with validation. No number below 0 is allowed:

```
public string Area
{
  get { return area; }
  set
  {
    if (value < 0) { area = 0; }
    else { area = value; }
  }
}
```