**Object-Oriented Programming** (OOP) – Writing programs that utilize classes and objects

**Four Pillars of OOP:**

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

**Abstraction -** Pulling related pieces of information and behavior into logical, reusable chunks

**Encapsulation** - The process of hiding grouped data behind a barrier (such as a class definition) and dictating how the data can be accessed and modified is known

**Access Modifiers:**

- We can use **access modifiers**, such as public and private, to control where the members of a class can be accessed:

* **public:** Any class can access a public member.
* **private:** A private member can only be accessed by code within the same class.

A computer code on a dark background

Description automatically generated

- For simplicity, we’ve been using public for all members so far. This allows code from the Main() method to access the members, even though Main() does not belong to our defined class. When we switch a field from public to private, it is no longer accessible from Main().

- Access modifiers apply to all class members, including fields and methods.

- We implement encapsulation using public and private to protect a class’s inner workings.

- Changed tree and age from public to private and created a new method within the same class that has to be called outside of the class in order to update those variables

A screenshot of a computer code

Description automatically generated A computer code with text

Description automatically generated A computer screen shot of text

Description automatically generated

**Properties:**

- Properties control access to a field, allowing us to validate values before they are set.  
- Common to name a property with the title-cased version of the fields name  
- Consist of two methods:

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

A computer code with text

Description automatically generated with medium confidence A screenshot of a computer program

Description automatically generated

A computer code with text

Description automatically generated with medium confidence A screen shot of a computer code

Description automatically generated

**Auto-Implemented Properties:**

- A shorthand for getter/setter that allows for more concise and readable code  
A computer code with text

Description automatically generated with medium confidence A screenshot of a computer code

Description automatically generated

In this form, we don’t have to write out the get() and set() methods or define a private size field. The field is automatically created in the background, so we only need to deal with the Size property.

**Get-Only Properties:**

- Can also allow programs to only get the value but not set it  
- Have two options:

- Setting Area without including set() results in an error regardless of where it is accessed. When set() is private, we get an error only outside the Shape class.

- Generally, we prefer a private set() because it allows other class [methods](https://www.codecademy.com/resources/docs/c-sharp/methods) to set the property.

1. Omit the set() method.
2. Make the set() method private.

A computer screen shot of text

Description automatically generated A screenshot of a computer code

Description automatically generated

A blue background with white and orange text

Description automatically generated

**Static Fields and Properties:**

- **Static members** are members that are associated with the class itself. They are defined using the static keyword, which should follow the access modifier if one is present.

Definition is a static property associated with the Book class. Within the class definition, static members can be referenced by their name alone, or using the class name and dot (.) notation. Outside the class definition, we must access static members using the class name and dot (.) notation. Note that we do not need to make an instance of Book to access its static member:

A computer screen shot of text

Description automatically generated

In the preceding code sample, we define two properties for the Book class. Title is a non-static property, so any instances of the Book class we create will each have its own Title property.

A computer screen shot of a program

Description automatically generated A screenshot of a computer screen

Description automatically generated

A computer code with text

Description automatically generated

Since Definition is an auto-implemented static property, the backing private field that is created automatically will also be static. If the related field and property are defined separately, they must match in being static or non-static.

If we tried to access a static member from an instance:

**Static Methods:**

- Can also create static methods to define behavior that applies to a class rather than its instances  
- Called from the class rather than an instance

A computer screen shot of code

Description automatically generated A computer screen shot of a computer code

Description automatically generated

- Static methods can only access other static members. They cannot access instance members:

A computer screen shot of text

Description automatically generated A computer screen shot of a error message

Description automatically generated

**Static Constructors:**

- An instance constructor is run each time a new instance of a class is created and typically handles setting values for instance fields and properties.  
- We can use a static constructor to handle setup for a class that only needs to be run once before the class is used. Typically, we use static constructors to set values for static fields and properties.  
- The static constructor of a class will only ever run once during the execution of a program, the first time either one of these events occurs:

* an object is made from the type
* another static member of the class is accessed

A screenshot of a computer code

Description automatically generated

The preceding code sample defines a Book class with a static constructor, which sets the value of the static Definition property.

A static constructor cannot have an access modifier or take parameters. A class may only have one static constructor, meaning they also cannot be overloaded. Static constructors cannot be called directly — they are only ever called automatically under the conditions specified before.

**Static Classes:**

- A static class cannot be instantiated and may only contain static members, so we should only create one if we are making a utility or library, like Math or Console.  
- We rarely create static classes of our own  
- These two common [classes](https://www.codecademy.com/resources/docs/c-sharp/classes) are static because they are just tools — they don’t need specific instances:

A screen shot of a computer code

Description automatically generated A screenshot of a computer code

Description automatically generated A computer screen shot of text

Description automatically generated