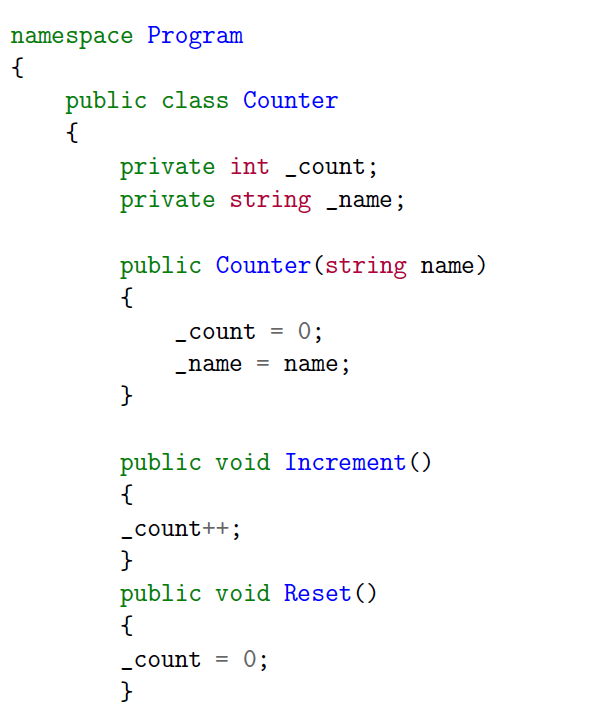
**6.2 The four principles of OOP**

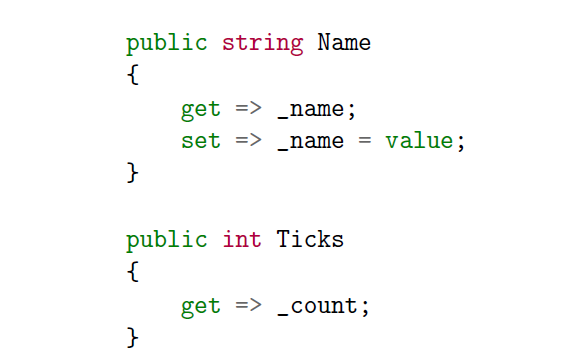
- The first principle: Encapsulation

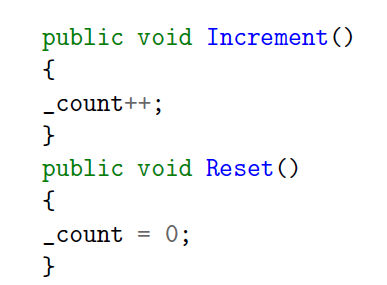
“Encapsulation is defined as the wrapping up of data and information under a single unit. It is the mechanism that binds together the data and the functions that manipulate them. In a different way, encapsulation is a protective shield that prevents the data from being accessed by the code outside this shield.” –geeksforgeeks

Encapsulation helps in making each class only knowing certain information by using private variables and using properties to set and get values. This helps the class only know what it needs to know, therefore, helping with data hiding as well as functionality. Not only that, it makes up-keeping code and debugging easier as if a certain variable in the class is causing a bug, we can quickly link it to the class.

The course has us practice encapsulation on our second week, where we use encapsulation in the Counter class (private count and name) where we could only access it from its class. Therefore a property was created to help us access the number, as well as methods such as Increment to += the count.

\_count and \_name was set as private.

properties were made to access \_count and \_name

methods were implemented to access and manipulate the ticks, because we cant set the ticks, Increment was made to manipulate it.

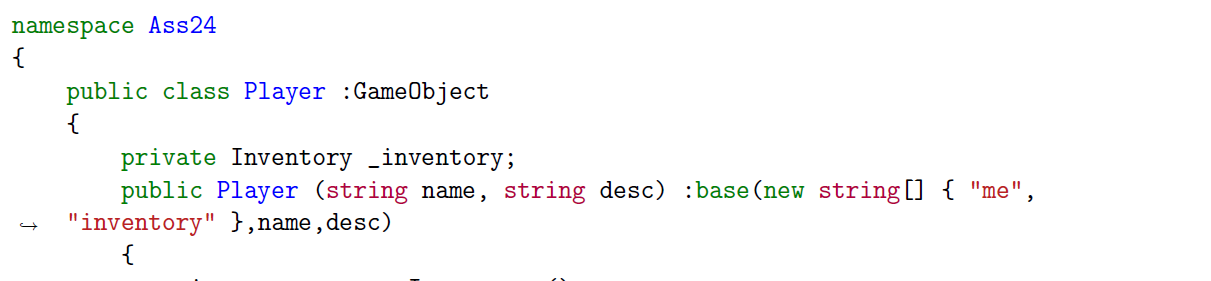
* Second principle: Inheritance

Inheritance is simple yet powerful.

“the ability to create a class that inherits attributes and behaviors from an existing class.”

-techopedia

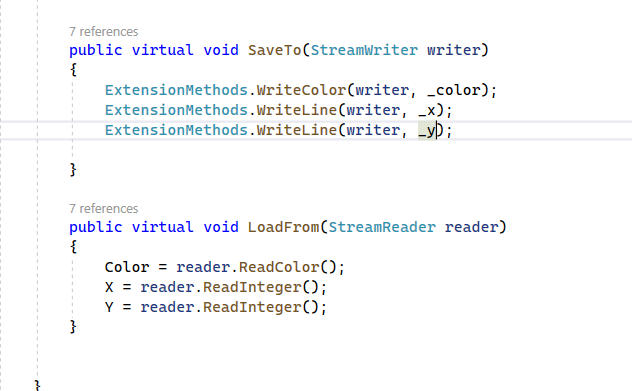
Inheritance is basically a class of parents and child. For example, an animal can eat and walk. Its children are cats and dogs, who inherit the ability to walk and eat. Cats can walk, eat, meow and dogs can bark, eat, walk. They all have the same characteristics as their parents however they have added characteristics exclusive to them.

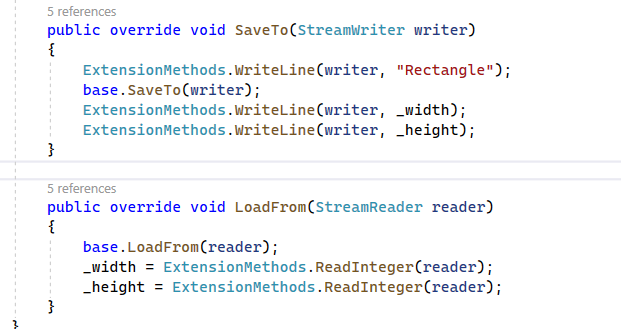


Player class inherits from the GameObject class. The base is called along with its values. The child class has access to every variable, methods,.. inside the parent class as long as it is not private.

This inheritance relationship enables the "Player" class to benefit from the functionality provided by the "GameObject" class while also allowing for additional customization or specialization in the "Player" class itself. This inheritance relationship enables the "Player" class to benefit from the functionality provided by the "GameObject" class while also allowing for additional customization or specialization in the "Player" class itself.

* Third Pillar Polymorphism:

Polymorphism is having multiple forms



This makes that the same class can have multiple instances of different methods. This promotes code reuse and flexibility.

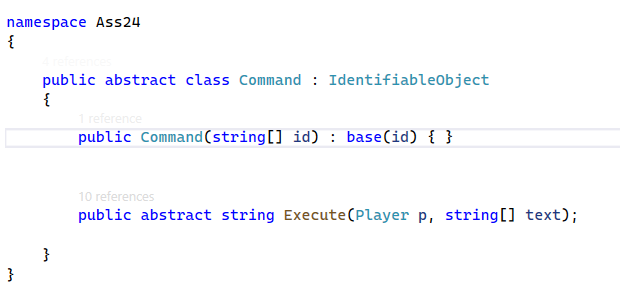
* Fourth Pillar Abstraction:

Data abstraction is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).

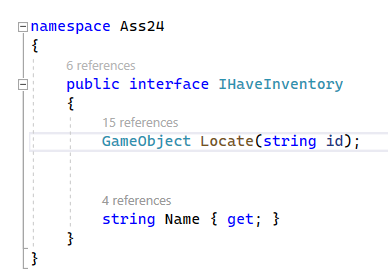
Abstraction with interface:

It is constructed with data security, hide certain details and show only the important parts.



In 6.1, we declared a class as abstract and left all the constructor and method empty. It is used to create a placeholder or a base class that can be inherited by other classes. This empty class serves as a starting point for creating more specialized classes.

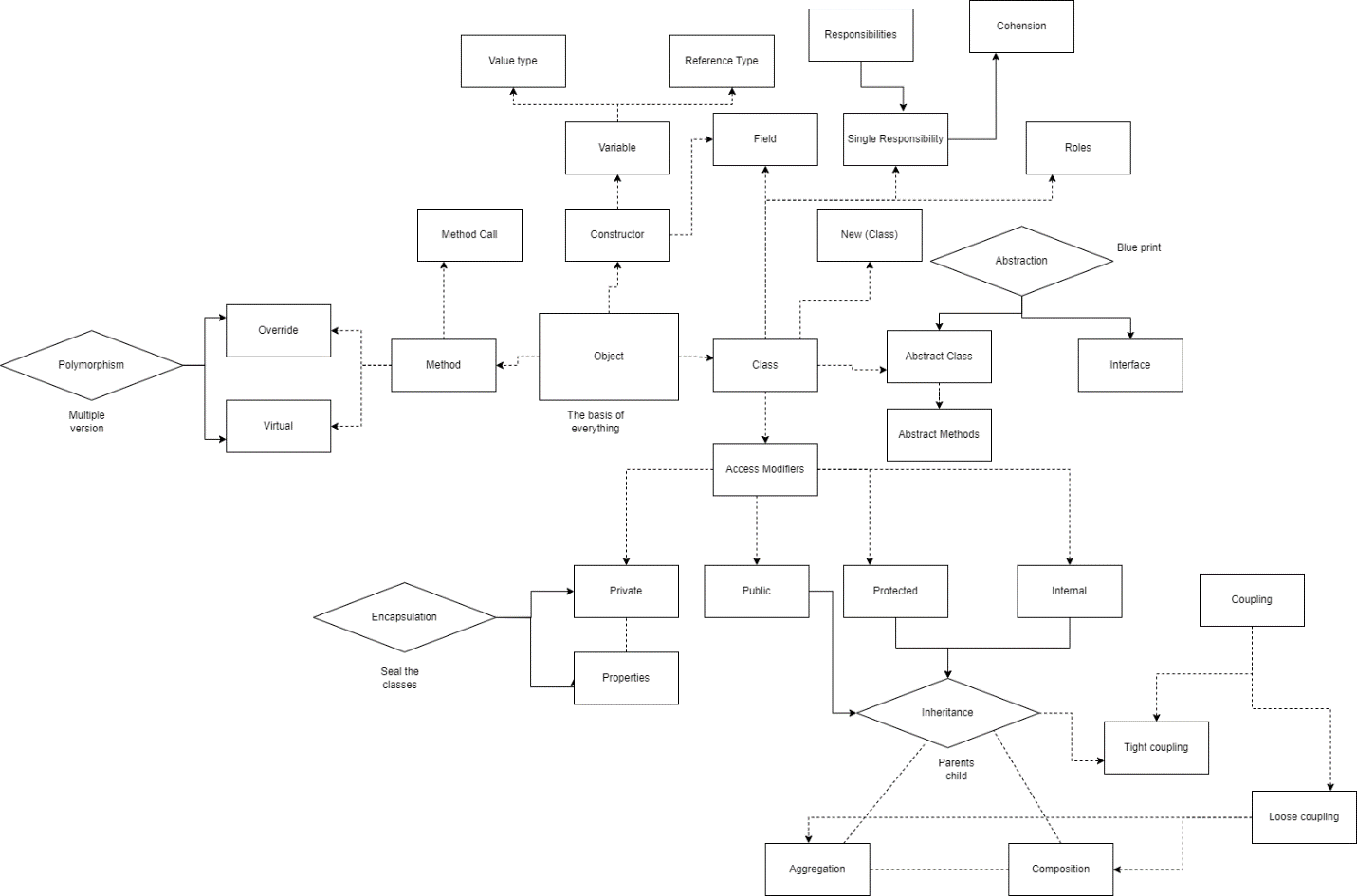
We can also implement abstraction using interface:



In 6.1 an interface was created to be like a set blue print that all classes must abide by.

Abstract classes are typically used for creating a base class for other classes to inherit from, while interfaces are used for defining a contract that classes must implement.

Concept Map:



Credit:

https://www.geeksforgeeks.org/c-sharp-encapsulation/

<https://www.techopedia.com/definition/27987/inheritance-c>