Object Oriented Programming

.NET

Object-Oriented Programming (OOP) *is based on the concept of "objects", which can contain data in the form of fields (attributes/properties), and code in the form of procedures (methods).*
In ***OOP****, computer programs are designed by making them out of objects (classes) that interact with one another.*

HTTPS://EN.WIKIPEDIA.ORG/WIKI/OBJECT-ORIENTED_PROGRAMMING

Four Pillars of OOP

https://www.linkedin.com/pulse/4-pillars-object-oriented-programming-pushkar-kumar#:~:text=

The four pillars of OOP

- Abstraction : The process of showing only essential/necessary features of an entity/object to the outside world and hide the other irrelevant information.
- Encapsulation : Wrapping data and member functions (Methods) together into a single unit (class). Encapsulation automatically achieves the concept of data hiding. This provides security to data by making variables private and allowing public methods access to the private variables.
- Inheritance : Creating a new class from an existing class template. A class (subclass) acquires the properties and behavior of a ‘base’ (‘super’) class.
- Polymorphism: "many forms". A subclass can inherit functionalities or behavior of its parent/base class and define its own unique behavior.

Abstraction

https://stackify.com/oop-concepts-c-sharp/#:~:text=

Abstraction is the process by which a developer separates the relevant data from the irrelevant details in order to simplify use.

Abstraction in daily life

- Apartment Building. We determine what the building is for by it's exterior or sign but don't know the specifics as to how the people live.
- Factory.
- Snail Mail.

Encapsulation

https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)

Encapsulation the restricting of direct access to abstracted data.

Encapsulation prevents unauthorized parties' direct access to the members of a class. Publicly accessible methods are generally provided in the class (so-called "getters" and "setters") to access the values.

R
RESTRICTED
UNDER 17 REQUIRES ACCOMPANYING PARENT OR ADULT GUARDIAN
VIOLENCE, GRISLEY IMAGES BRIEF STRONG LANGUAGE AND SOME NUDITY
RESTRICTED AREA
AUTHORIZED PERSONNEL ONLY

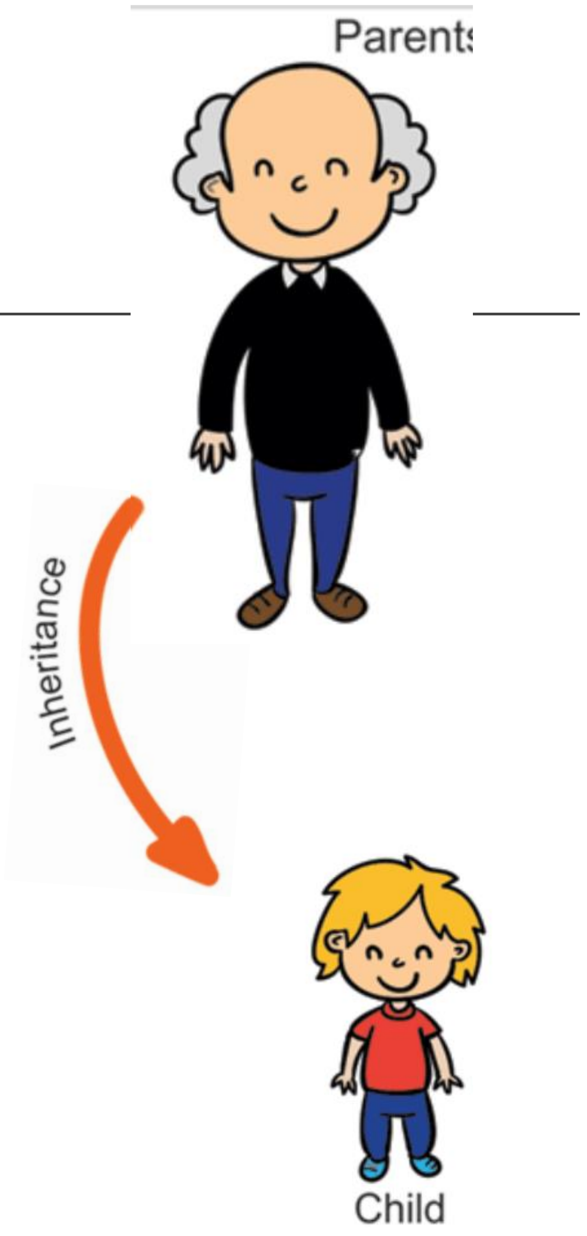
Inheritance

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance#:~:text=

Inheritance allows you to define a child class that reuses (inherits) the characteristics of a parent class.

The class that Inherits the members of the '***base' class*** is called the '***derived' class***.

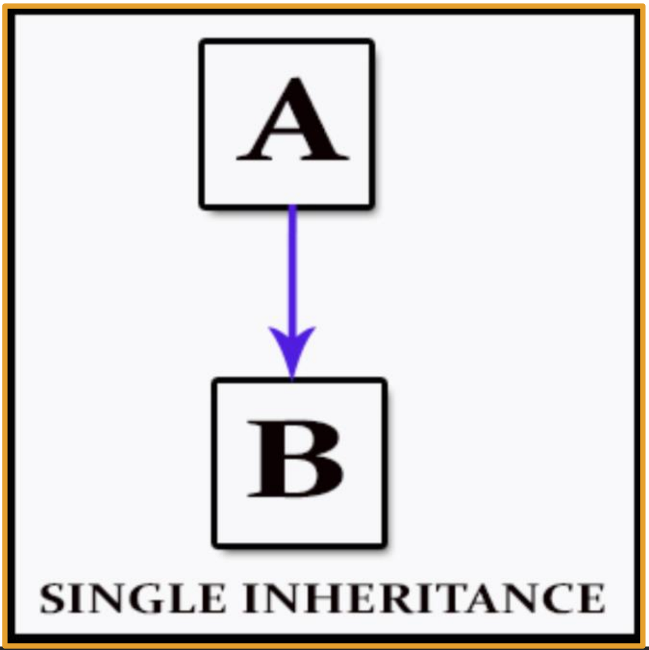
- structs, delegates, and enums do not support inheritance.



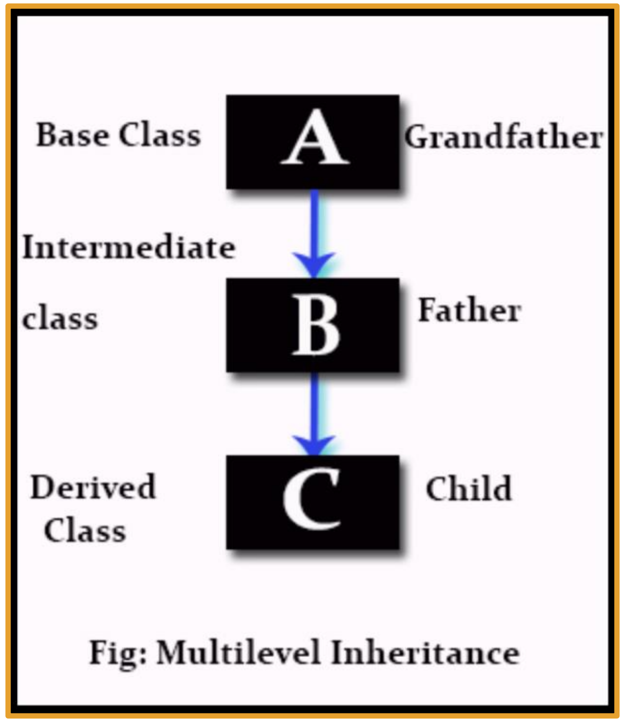
Inheritance - Types

https://stackify.com/oop-concepts-c-sharp/#:~:text=
https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)

Single inheritance(C#) - where subclasses inherit the features of one superclass. A class acquires the properties of another class.



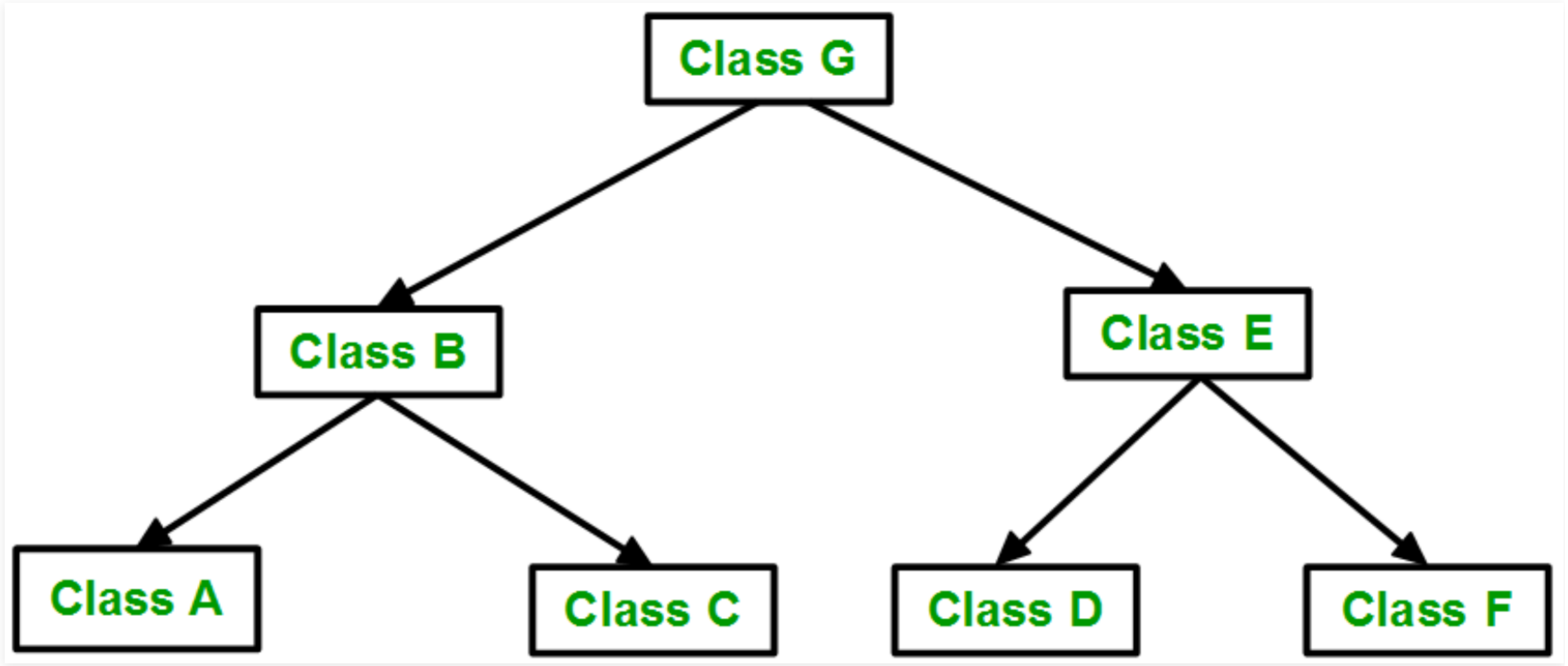
Multilevel inheritance(C#) - where a subclass is inherited from another subclass.



Inheritance - Types

https://stackify.com/oop-concepts-c-sharp/#:~:text=
https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)
https://www.geeksforgeeks.org/inheritance-in-c/

Hierarchical inheritance(C#) - where one class serves as a superclass (base class) for more than one sub class.

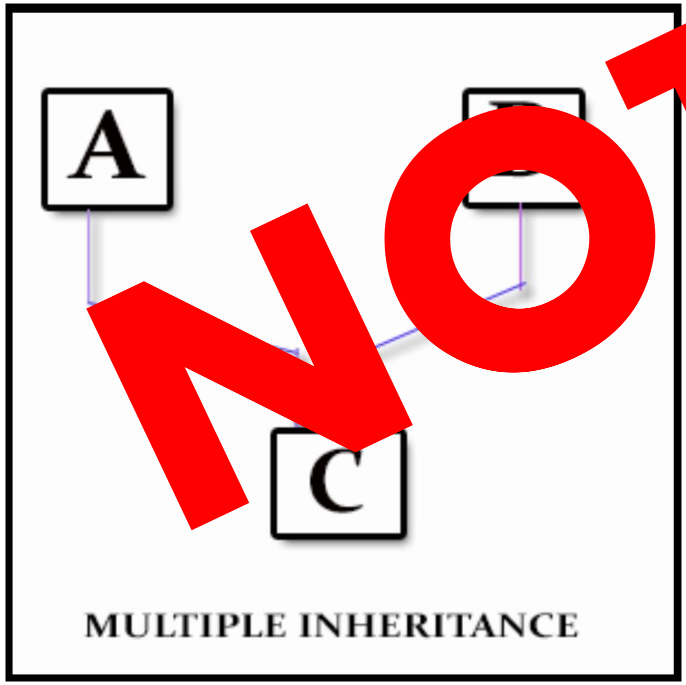


Inheritance - Types

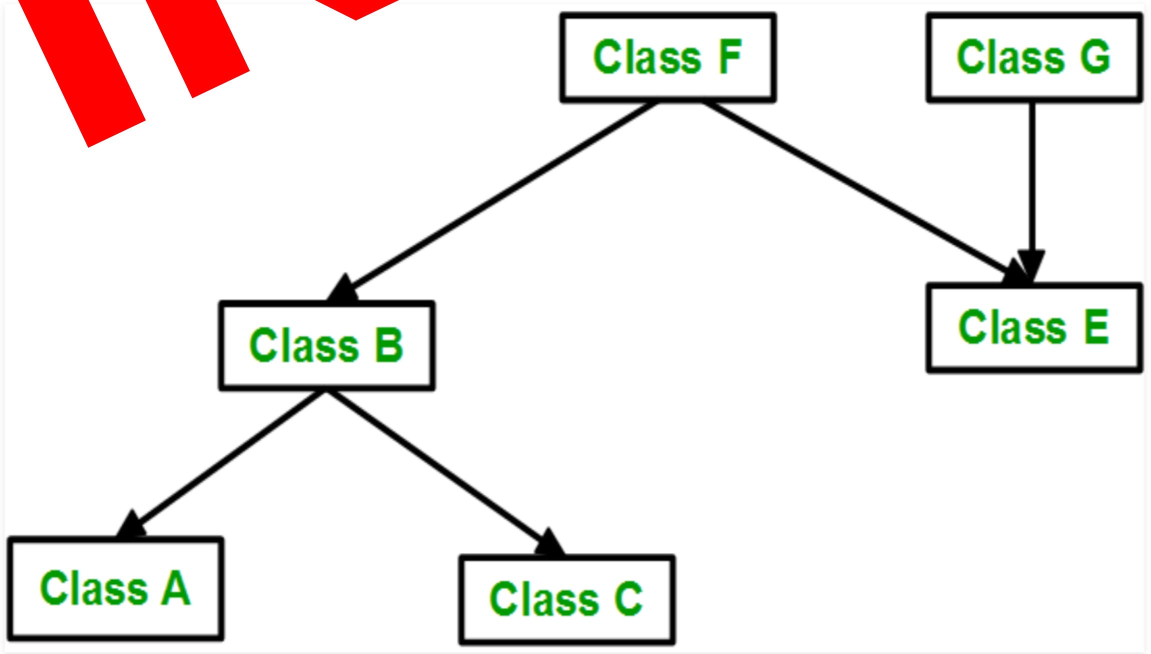
https://stackify.com/oop-concepts-c-sharp/#:~:text=
https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)
https://www.geeksforgeeks.org/inheritance-in-c/

NOT IN C#

Multiple inheritance(NOT IN C#) - one class can have more than one superclass and inherit features from all parent classes.



Hybrid inheritance(NOT IN C#) - a mix of two or more types of inheritance.



Inheritance and Access Modifiers

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance#:~:text=

Access Modifiers affect inheritance

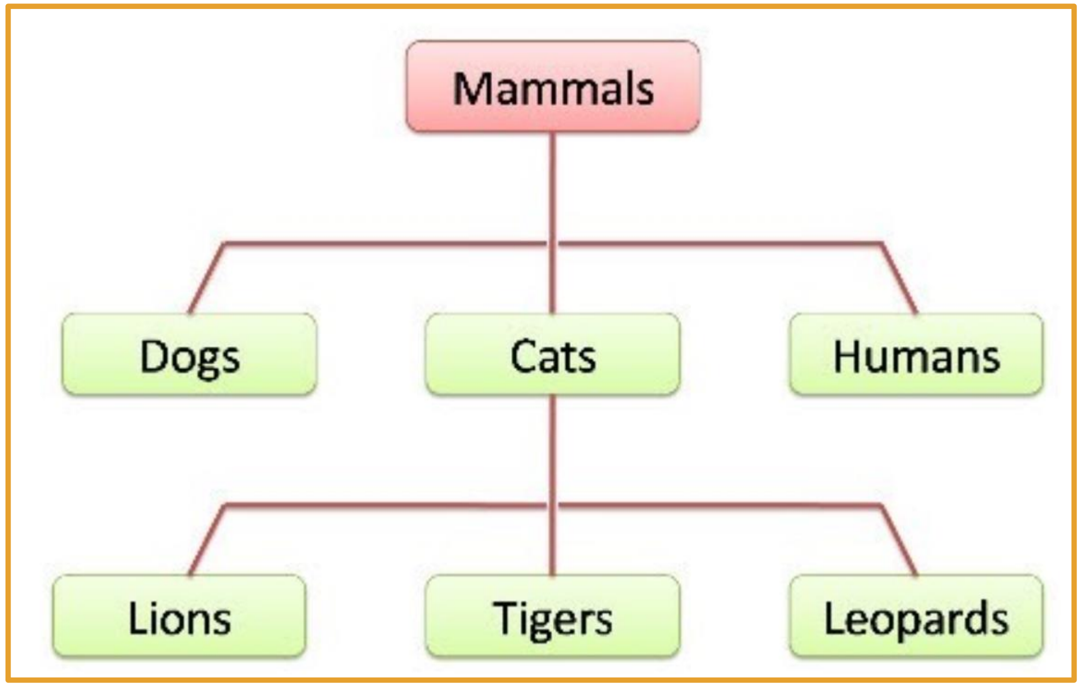
- Private -members are visible only in derived classes that are nested in their base class.
- Protected - visible only in ***derived*** classes.
- Internal - visible only in the same assembly as the ***base*** class.
- Public - visible in ***derived*** classes and are part of the ***derived*** class' public interface.
- Members of a ***base*** class that are NOT inherited by ***derived*** classes.
  - Static constructors – Which initialize the static data of a class.
  - Instance constructors – Each class must define its own constructors.

Inheritance – an ‘is a’ relationship

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance#:~:text=

Inheritance is used to express an "is a" relationship between a ***base*** class and one or more ***derived*** classes, where the ***derived*** class ‘is a’ specialized version of the ***base*** class.

An ‘is-a’ relationship based on inheritance is best applied to add additional members to the ***base*** class or that require additional functionality not present in the ***base*** class.



Polymorphism

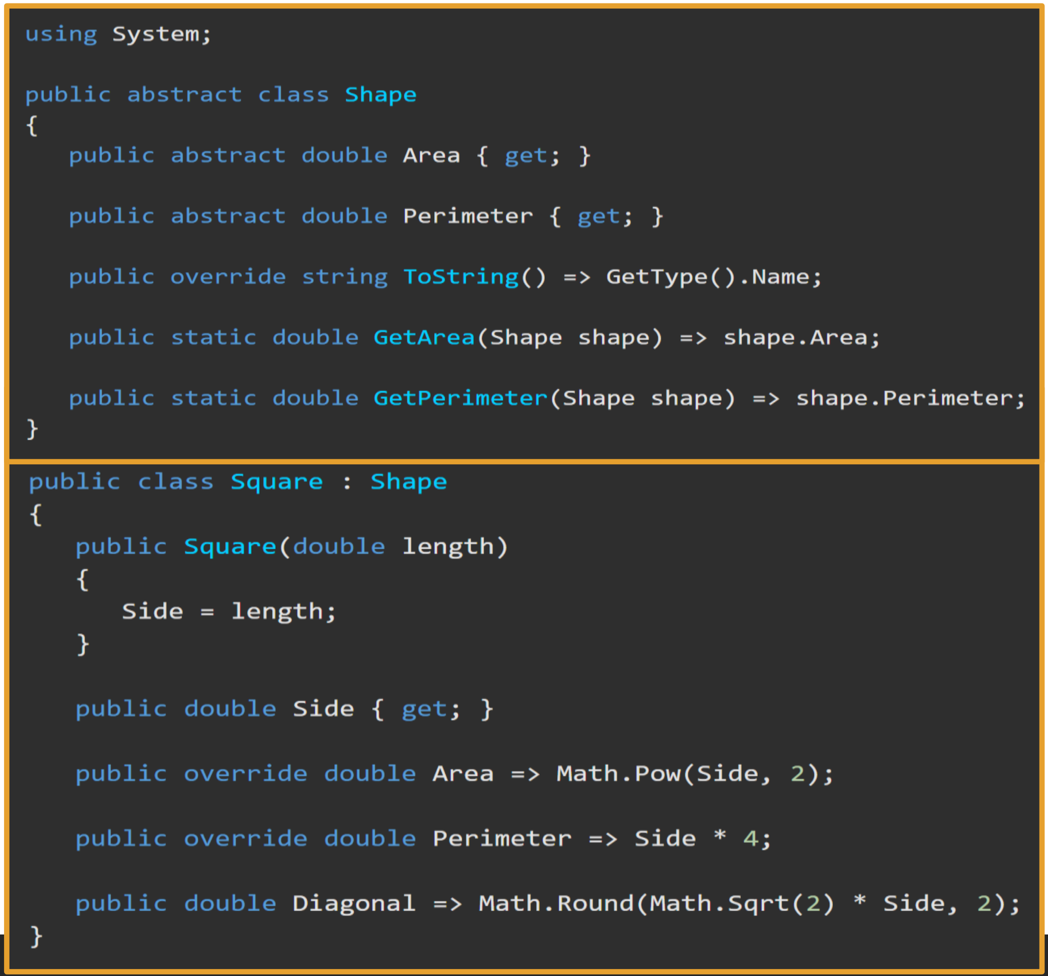
https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance#designing-the-base-class-and-derived-classes

Polymorphism is when each ***derived class*** implements the same methods but in different ways.

If a ***base class*** member is marked ***abstract***, it must be defined in the ***derived class***.

Only ***virtual base class*** members may be ***overridden***.

Only ***derived*** class members using the ***override*** keyword may implement an alternative definition of the ***virtual base class*** member.



```
using System;

public abstract class Shape
{
   public abstract double Area { get; }

   public abstract double Perimeter { get; }

   public override string ToString() => GetType().Name;

   public static double GetArea(Shape shape) => shape.Area;

   public static double GetPerimeter(Shape shape) => shape.Perimeter;
}
```

```
public class Square : Shape
{
   public Square(double length)
   {
      Side = length;
   }

   public double Side { get; }

   public override double Area => Math.Pow(Side, 2);

   public override double Perimeter => Side * 4;

   public double Diagonal => Math.Round(Math.Sqrt(2) * Side, 2);
}
```

Activity

Complete the implementation of the Publication => Book program.