Clases y objetos: son loselmentos base para programar, clases nos ayudan a definir propiedad o plantillas para crear los objetos

Encapsulamiento: oculta los valores reales o el dettale de los valores de las características de nuestras clases

Herencia: es poder reutilizar código, pasar propiedades y metdos de una clase ya existente a tora

Polimorfismo: definir diferente comportameintes para diferetnes métodos

Abstracción: permite omitr los detalles de las implementaciones que hacemos en nuestra clase

Programación orientada a objetos: el objetivo es extraer elementos de la realidad o los algoritmos en nuestro código, basado en clases y objetos

Paradigma de la programación: formas o estilos en los que podemos programar y estructurar nuestro código

Paradigmas importantes estos son los más populares

Programación funcional

Programación estructurada

Programación reactiva

Programación orientada a aspectos

Programación orientada a objetos

C# es multiparadigma: lo común aquí es mezclar múltiples paradigmas al estructurar código

Para crear clases

```
Class nombredelaclase
{
        String(tipo de dato) color(nombredeldato);
}
```

Double: me permite manejar decimales

Short: es un entero corto

Para crear objetos hay dos formas

Var representa cualquier tipo de dato

Apuntador elcelular = new Apuntador(); especificamoes le tipo, le ponemos un nombre lo mas descriptivo posible seguido denew que es lo que nos permite crear un objeto a partier de una clase y por ulitmo el nombre de la clase y cerrar con ;

Asignar valores aun objeto

Clase: es el elemento principal de un pryecto de POO, se basa en propiedades que representan las características del objeto y métodos que definen el comportamiento, se pueden defirnr como plantilla base para crear objetos

Los objetos son instancias de las clases, ya que siempre estos estarán haciendo eferancia a la calses de donde fieron creados

Las clases tienen dos componentes principales: las propiedades y los métodos

Los objetos los vamos utilizarlos para guardar datos

Enumeraciones en csharp nos permiten asignar valores específicos o crear una lista de estos y forzar a quien use la clase que asigne los valores ya aquí definidos

Métodos: función que realiza una acción dentro de la clase

Void: la función ejecutara una función, pero no devolverá un valor si no que lo utilizara de forma interna

Una propiedad normalmente se le asigna un valor con =

Un método se toma el nombre y se utilizan los paréntesis para realizar su ejecución

Tabla comparativa

Clase	Estructura	Registro
Referencia	Valor	Valor o referencia
Grandes	Pequeñas	Pequeñas
Valores y comportamientos	Enfocada a valores	Enfocada a valores inmutables

Modificadores de acceso: lista de modificadores public, protected, internal, private, file

Public y private son los que mas se utilizan private evita que cualquierq que este fuera de nuestro contexto de clase pueda utilizar algún método o propiedad y public hace lo contrario

Protected: el acceso esta limitado a la clase contenedora o a los tipos derivados de la clase contenedora

Internal: el acceso está limitado al ensamblado actual

Herencia

Se remite a las clases, cuando se usa herencia una propiedad y un método pasaran de la clase primaria a la clase hija y en estase pueden crear métodos adicionales

Sintaxis en C#: public class ClaseHija:ClasePadre

En C# solo podemos heredar de una sola clase

Abstracción

Si queremos crear un método abstracto es decir que no tiene lógica, cuerpo o implementación en particular, entonces la clase tiene que ser abstracta

Polimorfismo

Es usar un mismo método, pero con diferentes funciones

Interfaz

Es un contrato que nos ayuda a garantizar una estructura dentro de la clase y poder utilizarla en diferente parte de nuestro código, permite elementos abstractos e implementaciones por defecto, ayudan a desacoplar (que no dependan una de la otra si no de componentes abstractos) el código, C# soporta implementación de múltiples interfaces

Características

Normalmente se nombran con **I**nombre

Se utilizan en casi todos los patrones de diseño para .NET

Podemos implementar generics para utilizar la misa interfaz en diferentes escenarios, Ejemplo IList<T>

Comparativa

Clase abstracta	Interfaz
Solo permite una herencia	Permite múltiples implementaciones
Permite tener implementaciones	No es posible implementar (se puede desde C# 8.0)
Recomendada para reutilizar código y lógica	Recomendada para implementar patrones de diseño y la inyección de dependencias
Menos usada	Altamente usada