C3 es un lenguaje que se caracteriza por su sintaxis, porque es fuertemente tipado, esta orientado a objetos, case sensitive, tu una gran importancia, la ide en la que se utiliza es  de visual studiom editor de código, azure devops y app center o visual studio

Aplicaciones en c#

Existen 7 categorias de software de computadoras que presentan retos continmuos para los ingenieros, desoftware, estas son : software de sistemas, software de aplicaciones , software científicos y de ingenerias , software empotrado, software de línea de productos,  aplicaciones basadas en la web y software de inteligencia artificial

Otras clasificaciones o cosas que puedes hacer en c# :

- Aplicaciones de escritorio
- Web aplicaciones
- Servidores
- Programas para moviles
- Programas de juegos
- Soluciones de empresas
- Bases de datos

Un namespace permite agrupar un conjunto de clases, esto es útil ya que dentro de un mismo paquete no pueden existir clases del mismo nombre.

C# es un lenguaje fuertemente tipado, lo que exige el casteo de tipos.

Clases y objetos :

En c# las clases son plantillas que se utilizan para crear objetos. Un objeto es una instancia de una clase. Las clases se definen utilizando la palabra clave class, seguida del nombre de la clase y un conjunto de llaves que contiene las propiedades y métodos de la clase

Metods y constructores

Los métodos son funciones que se definen en una clase y se utilizan para realizar acciones en objetos de esa clase. Los constructores son métodos especiales que se utilizan para inicializar objetos de una clase.

Modificadores de acceso

En c# los modificadores de acceso son palabras clave que se utilizan para determinar el nivel de acceso a las propiedades y métodos de una clase. Los modifcadores de acceso son public, private, protec, internal y protected internal

Public: puede obtener acceso al tipo o miembro cualquier otro código del mismo ensamblado o de otro ensamblado que haga referencia a este.

Protected: puede obtener acceso al tipo o miembro solo dentro de su propia clase o en las clases derivadas

Internal: puede obtener acceso al tipo o miembro solo dentro del mismo ensamblado

Private puede obtener acceso al tipo o miembro solo dentro de la misma clase o estructura

Protected internal:  puede obtener acceso al tipo o miembro dentro del mismo ensablado  en las clases derivadas fuera del ensamblado

Un ensamblado en c# es un archivo que contiene código compilado, recursos y metadatos que describen contenido del ensamblados. Los ensamblados se utilizan para agrupar y disfrutar código de manera lógica y reutilizable

Una aplicación o una dll son eke,plos de nsamblados en c#

El modificador de acceso “internal” se utiliza para establecer que un tipo o miembro solo puede ser accedido dentro del mismo ensamblado

El modificador de acceso protected internal es una combinación de los modifcadores de acceso protected e internal. Un miembro con el modificador protected internal es accesible desde el ensamblado actual y desde cualquier tipo derivado de la clase contenedora.

Herencia: la herencia es un concepto importante en la programación orientada a objetos que permite a las clases heredar propiedades y métodos de otras clases. La clase que se hereda se llama clase base o clase padre, y la clase que se hereda se llama clase derivada o clase hija

Override de métodos

El override de métodos es un concepto importante en la programación orientada a objetos que permite a las clases derivadas reemplzar la implementación de un método de la clase base.

Modifcador base: la palabra clave “base” se utiliza para acceder a los miembros de la clase base desde una clase derivada. Solo se permite el acceso a la clase base en un constructor, en un meotod de instancia y un descriptor de acceso de propiedad de instancia. El uso de la palabra clase “ base” desde dentro de un método estático dará un error

EN C#  UNA INTERFAZ es una estructura que se compone de métodos y constantes

Los métodos de una interfaz son declarados pero no implementados

La interfaz indica que métodos debe obligatoriamente implementar una clase

Porque es tan importante ¿ porque le da flexibilidad a tu apliacion. En lugar de que tu código solo pueda trabajar con una subclase especifica, como baseballplayer, puedes trabajar con algo mas genérico, como athlete. Esto significa que tu código funcionara con cualquier subclase de athlete, como hockeyplayer o tennisplayer e incluso con subclases que aun no han sido siseladas

Inyección de depencias: la inyección de dependencia es un conjunto de principios y patrones de diseño de software que nos permite desarrollar código poco acoplado

La DI no es un objetivo final en si mismo, sino un meido para lograr otros objetivos

La di permite el acomplamiento débil, lo que hace que el código sea mas mantenible

En el desarrollo de software, estamos aprendiendo aun como implementar una buena arquitectura, pero otras profesiones, como la construcción, ya han encontrado soluciones a problemas similares.

El acomplamineto estrecho es como conectar directamente un secador de pelo a la pared, lo que dificulta la modificación de uno sin afectar al otro

El acomplamiento débil loosely coupled se parece mas ala conexión de electrodomésticos con enchhufed y tomas de corrientes. Esto permite conectar dispositivos de diferentes tipos de forma flexible

Los enchufes y tomas de corriente pueden compararse a patrones de diseño de software conocidos, como el principio de sustitución de liskov

La flexibilidad de las tomas de corrientes y enchufes permite adaptarse a necesidades futuras que no se habían previsto original

La utilización de enchuifes y tomas de corrientes es un ejemplo de como se pueden aplicar principios de diseño de diferentes campos

- Inyección de dependencias en el constructor
- Inyección de dependencias en métodos

Encapsulamiento:

- La encapsulación es importante para prevenir la duplicación de código.
- La encapsulación también ayuda a proteger tus clases de cambios innecesarios.
- Si tienes comportamientos en tu aplicación que probablemente cambiarán, debes separarlos de las partes que no cambiarán con tanta frecuencia.
- La encapsulación implica separar y ocultar la complejidad de una clase, y exponer solo la interfaz necesaria para trabajar con ella.
- Al encapsular correctamente, puedes hacer que tu código sea más mantenible y escalable en el futuro.
- Parece que la clase Pintor tiene dos métodos que son bastante estables, pero el método paint() va a variar mucho en su implementación. Así que encapsulemos lo que varía y movamos la implementación de cómo un pintor pinta fuera de la clase Pintor.

Cambio:

- El cambio es la única constante en el software.
- El software mal diseñado se desmorona al primer signo de cambio, mientras que el gran software puede cambiar fácilmente.
- La forma más fácil de hacer que el software sea resistente al cambio es asegurarse de que cada clase tenga solo una razón para cambiar.
- Al reducir el número de cosas en una clase que pueden causar que cambie, se minimiza la posibilidad de que la clase tenga que cambiar.

- Cada clase debe tener una sola responsabilidad y no tener múltiples razones para cambiar.
- Si una clase tiene múltiples razones para cambiar, su diseño puede ser mejorado para reducir la dependencia de la clase.
- Cuando ves una clase que tiene más de una razón para cambiar, probablemente está tratando de hacer demasiadas cosas. Intenta dividir la funcionalidad en varias clases, donde cada clase individual hace solo una cosa, y por lo tanto solo tiene una razón para cambiar.