Introducing C# and .NET

C# is a general-purpose, type-safe, object-oriented programming language. The goal of the language is programmer productivity. To this end, C# balances simplicity, expressiveness, and performance. The chief architect of the language since its first version is Anders Hejlsberg (creator of Turbo Pascal and architect of Delphi). The C# language is platform neutral and works with a range of platform-specific runtimes.

ترجمه پاراگراف

#Cیک زبان برنامه‌نویسی همه‌منظوره، ایمن از نظر نوع (type-safe) و شیءگرا (object-oriented) است. هدف اصلی این زبان، افزایش بهره‌وری برنامه‌نویس‌هاست. برای رسیدن به این هدف، #Cبین سادگی، بیان‌پذیری (expressiveness) و کارایی (performance) تعادل ایجاد می‌کند.
معمار اصلی این زبان از نسخه اول تاکنون آندرس هایلسبرگ بوده است) خالق Turbo Pascal و معمار Delphi).
زبان #C مستقل از پلتفرم است و می‌تواند با مجموعه‌ای از runtimeهای خاص هر پلتفرم کار کند.

توضیحات تکمیلی و نکات ارائه

1. General-purpose:
   یعنی #C محدود به یک نوع نرم‌افزار خاص نیست؛ میشه باهاش وب‌اپلیکیشن، اپلیکیشن موبایل، دسکتاپ، بازی، سیستم‌های توزیع‌شده و حتی هوش مصنوعی نوشت.
2. Type-safe:
   - امنیت در نوع داده‌ها.
   - یعنی مثلاً نمی‌تونی به اشتباه یک عدد صحیح رو مثل یک رشته (string) استفاده کنی بدون تبدیل درست.
   - این باعث میشه خطاهای برنامه کمتر باشه.
3. Object-Oriented:
   - همه‌چیز حول محور کلاس و شیء می‌چرخه.
   - پشتیبانی از مفاهیمی مثل ارث‌بری، پلی‌مورفیسم، کپسوله‌سازی.
4. Programmer Productivity:
   - #C کمک میکنه برنامه‌نویس سریع‌تر، راحت‌تر و با خطای کمتر کد بزنه.
   - IntelliSense، کتابخانه‌های آماده، و syntax ساده → همه در همین راستا طراحی شدن.
5. Platform Neutral:
   - #Cخودش مستقل از پلتفرمه.
   - چیزی که تغییر می‌کنهruntime ها هستن) مثلاً .NET Framework روی ویندوز، .NET 8 روی همه سیستم‌ها، Unity runtimeبرای بازی‌ها.(
   - یعنی یک زبان، اما چند محیط اجرا.

Object Orientation

C# is a rich implementation of the object-orientation paradigm, which includes encapsulation, inheritance, and polymorphism. Encapsulation means creating a boundary around an object to separate its external (public) behavior from its inter nal (private) implementation details. Following are the distinctive features of C# from an object-oriented perspective: Unified type system The fundamental building block in C# is an encapsulated unit of data and func tions called a type. C# has a unified type system in which all types ultimately share a common base type. This means that all types, whether they represent business objects or are primitive types such as numbers, share the same basic functionality. For example, an instance of any type can be converted to a string by calling its ToString method. Classes and interfaces In a traditional object-oriented paradigm, the only kind of type is a class. In C#, there are several other kinds of types, one of which is an interface. An interface is like a class that cannot hold data. This means that it can define only behavior (and not state), which allows for multiple inheritance as well as a separation between specification and implementation. 1 Properties, methods, and events In the pure object-oriented paradigm, all functions are methods. In C#, meth ods are only one kind of function member, which also includes properties and events (there are others, too). Properties are function members that encapsulate a piece of an object's state such as a button's color or a label's text. Events are function members that simplify acting on object state changes. Although C# is primarily an object-oriented language, it also borrows from the functional programming paradigm, specifically: Functions can be treated as values Using delegates, C# allows functions to be passed as values to and from other functions. C# supports patterns for purity Core to functional programming is avoiding the use of variables whose values change, in favor of declarative patterns. C# has key features to help with those patterns, including the ability to write unnamed functions on the fly that "capture" variables (lambda expressions), and the ability to perform list or reactive programming via query expressions. C# also provides records, which make it easy to write immutable (read-only) types

ترجمه پاراگراف

شیءگرایی
#Cپیاده‌سازی غنی‌ای از پارادایم شیءگرایی ارائه می‌دهد که شامل کپسوله‌سازی(encapsulation) ، ارث‌بری (inheritance) و چندریختی (polymorphism) است.
کپسوله‌سازی یعنی ایجاد یک مرز به دور یک شیء برای جداسازی رفتار بیرونی (عمومی) آن از جزئیات پیاده‌سازی درونی (خصوصی).

ویژگی‌های متمایز #C از دیدگاه شیءگرایی:

سیستم نوع یکپارچه:(Unified type system)
واحد بنیادی در#C ، یک واحد کپسوله‌شده از داده و توابع است که به آن typeگفته می‌شود #C .دارای یک سیستم نوع یکپارچه است که در آن همه‌ی انواع (types) در نهایت یک پایه مشترک دارند. این یعنی همه انواع، چه نشان‌دهنده‌ی اشیای تجاری باشند یا انواع ابتدایی مانند عددها، عملکردهای پایه‌ی یکسانی دارند. برای مثال، نمونه‌ی هر نوع را می‌توان با فراخوانی متد ToString به رشته (string) تبدیل کرد.

کلاس‌ها و اینترفیس‌ها:(Classes and interfaces)
در پارادایم سنتی شیءگرایی، تنها نوع موجود کلاس است. در#C ، انواع دیگری هم وجود دارند که یکی

از آن‌ها interfaceاست. اینترفیس شبیه یک کلاس است، اما نمی‌تواند داده نگه دارد. یعنی فقط می‌تواند رفتار (behavior) تعریف کند، نه حالت .(state) این ویژگی امکان چند ارث‌بری و همچنین جداسازی مشخصات (specification) از پیاده‌سازی (implementation) را فراهم می‌کند.

ویژگی‌ها(Properties) ، متدها (Methods) و رویدادها:(Events)
در پارادایم خالص شیءگرایی، تمام توابع به شکل متد هستند. در#C ، متدها فقط یکی از انواع اعضای تابعی (function members) هستند. علاوه بر آن‌ها، ویژگی‌ها (properties) و رویدادها (events) هم وجود دارند (و موارد دیگر نیز). ویژگی‌ها اعضایی هستند که بخشی از حالت یک شیء را کپسوله می‌کنند، مثل رنگ یک دکمه یا متن یک لیبل. رویدادها اعضایی هستند که واکنش به تغییر حالت یک شیء را ساده می‌کنند.

گرچه #C عمدتاً یک زبان شیءگراست، اما از پارادایم برنامه‌نویسی تابعی (functional programming) هم الهام گرفته است، به طور خاص:

- توابع به‌عنوان مقدار:
با استفاده ازdelegates ، #Cاجازه می‌دهد توابع مثل مقادیر به توابع دیگر ارسال یا از آن‌ها بازگردانده شوند.
- الگوهای خلوص:(Purity patterns)
اساس برنامه‌نویسی تابعی اجتناب از متغیرهایی است که مقدارشان تغییر می‌کند و به‌جای آن از الگوهای اعلامی (declarative patterns) استفاده می‌شود.
#Cویژگی‌های کلیدی‌ای برای پشتیبانی از این الگوها دارد، از جمله امکان نوشتن توابع بی‌نام در لحظه که متغیرها را "capture" می‌کنند (lambda expressions) و امکان برنامه‌نویسی لیستی یا واکنشی (reactive) از طریق.query expressions
همچنین #C قابلیت recordsرا ارائه می‌دهد که نوشتن انواع تغییرناپذیر (immutable) یا فقط خواندنی را ساده می‌کند.