Type Safety

C# is primarily a type-safe language, meaning that instances of types can interact only through protocols they define, thereby ensuring each type's internal consis tency. For instance, C# prevents you from interacting with a string type as though it were an integer type. More specifically, C# supports static typing, meaning that the language enforces type safety at compile time. This is in addition to type safety being enforced at runtime. Static typing eliminates a large class of errors before a program is even run. It shifts the burden away from runtime unit tests onto the compiler to verify that all the types in a program fit together correctly. This makes large programs much easier to manage, more predictable, and more robust. Furthermore, static typing allows tools such as IntelliSense in Visual Studio to help you write a program because it knows for a given variable what type it is, and hence what methods you can call on that variable. Such tools can also identify everywhere in your program that a variable, type, or method is used, allowing for reliable refactoring.

ترجمه پاراگراف

#Cدر درجه‌ی اول یک زبان ایمن از نظر نوع (type-safe) است؛ یعنی نمونه‌های انواع مختلف تنها از طریق پروتکل‌هایی که خودشان تعریف می‌کنند می‌توانند با هم تعامل داشته باشند. این موضوع تضمین می‌کند که انسجام داخلی هر نوع حفظ شود. برای مثال، #Cاجازه نمی‌دهد با یک نوع رشته (string) طوری رفتار کنید که انگار یک عدد صحیح (integer) است.

به طور دقیق‌تر، #Cاز نوع‌گذاری ایستا (static typing) پشتیبانی می‌کند؛ به این معنا که زبان در زمان کامپایل ایمنی نوع را بررسی می‌کند. این موضوع علاوه بر بررسی ایمنی نوع در زمان اجراست.
نوع‌گذاری ایستا یک دسته بزرگ از خطاها را قبل از اجرای برنامه حذف می‌کند. در واقع بار بررسی را از تست‌های زمان اجرا به کامپایلر منتقل می‌کند تا اطمینان حاصل شود همه انواع در یک برنامه به‌درستی با هم سازگارند.

این ویژگی مدیریت برنامه‌های بزرگ را بسیار آسان‌تر، قابل پیش‌بینی‌تر و مقاوم‌تر می‌کند. افزون بر این، نوع‌گذاری ایستا به ابزارهایی مانند IntelliSense در Visual Studio کمک می‌کند برنامه‌نویسی را ساده‌تر کنند، چون دقیقاً می‌دانند که یک متغیر از چه نوعی است و بنابراین چه متدهایی را می‌توان روی آن فراخوانی کرد. چنین ابزارهایی همچنین می‌توانند همه جاهایی که یک متغیر، نوع یا متد در برنامه استفاده شده را شناسایی کنند، و این موضوع بازآرایی (refactoring) کد را قابل اعتماد می‌سازد.

#Cهمچنین اجازه می‌دهد بخش‌هایی از کد به‌طور پویا نوع‌گذاری شوند... (dynamically typed)

#Cهمچنین اجازه می‌دهد بخش‌هایی از کد شما با استفاده از کلیدواژه‌ی `dynamic` به‌طور پویا نوع‌گذاری شوند. با این حال، #C همچنان عمدتاً یک زبان با **نوع‌گذاری ایستا (statically typed)** باقی می‌ماند.

توضیحات تکمیلی و نکات ارائه

1. کلیدواژه :dynamic
   - وقتی یک متغیر رو با dynamicتعریف کنی، بررسی نوعش به زمان اجرا منتقل میشه.

- یعنی کامپایلر دیگه چک نمی‌کنه که آیا فلان متد یا پراپرتی روی اون متغیر وجود داره یا نه.
- اگر در زمان اجرا وجود نداشته باشه → خطا می‌گیری.(Runtime Error)

2. کاربردهای :dynamic
   - کار با APIهای انعطاف‌پذیر مثلJSON ، XML، یا داده‌هایی که نوعشون در زمان کامپایل معلوم نیست.
   - تعامل با کتابخانه‌های COM یا زبان‌های اسکریپتی) مثل Python از طریق.(interop
   - سناریوهایی که static typing بیش از حد محدود میشه.
3. چرا #C همچنان statically typed باقی مونده؟
   - چون static typing امنیت و قابلیت اطمینان بیشتری میده.
   - dynamicفقط برای جاهایی هست که انعطاف لازم داریم.
   - به همین دلیل استفاده از dynamic باید محدود باشه.

جمع‌بندی ارائه برای این قسمت:
"#Cامکان dynamic typing رو هم فراهم کرده تا در شرایط خاص مثل کار با JSON یاAPI های ناشناخته راحت‌تر باشیم. اما همچنان ماهیت اصلی زبان استاتیک باقی مونده، چون static typing پایه‌ی اطمینان، خطای کمتر و بهره‌وری بالاتر در پروژه‌های بزرگ رو فراهم می‌کنه".

C# is also called a strongly typed language because its type rules are strictly enforced (whether statically or at runtime). For instance, you cannot call a function that's designed to accept an integer with a floating-point number, unless you first explicitly convert the floating-point number to an integer. This helps prevent mistakes.

ترجمه پاراگراف

#Cهمچنین یک زبان) strongly typedشدیداً نوع‌گذاری‌شده (نامیده می‌شود، زیرا قوانین نوع در آن به‌طور سخت‌گیرانه اجرا می‌شوند (چه در زمان کامپایل و چه در زمان اجرا).
برای مثال، شما نمی‌توانید تابعی را که برای دریافت یک عدد صحیح (integer) طراحی شده با یک عدد اعشاری (floating-point number) صدا بزنید، مگر اینکه ابتدا عدد اعشاری را به‌طور صریح (explicitly) به عدد صحیح تبدیل کنید. این ویژگی به جلوگیری از خطاها کمک می‌کند.

توضیحات تکمیلی و نکات ارائه

1. Strongly Typedیعنی چی؟
   - زبان اجازه نمی‌ده به‌طور ضمنی انواع ناسازگار با هم قاطی بشن.
   - باید تبدیل نوع (casting) رو خودت صریحاً انجام بدی.

- مزیت:Strong Typing

  - خطاهای ناشی از تبدیل‌های ناخواسته یا اشتباه کم میشه.
  - برنامه‌نویس مجبور میشه آگاهانه تغییر نوع بده.
  - نتیجه: کد قابل‌اعتمادتر و پیش‌بینی‌پذیرتر.

- تفاوت با زبان‌های:Weakly Typed

  - مثلاً در JavaScript یا Python می‌تونی یک متغیر رو بدون دردسر به شکل‌های مختلف استفاده کنی، حتی اگه ناسازگار باشه (که گاهی خطاهای عجیب تولید می‌کنه).
  - در C# این اتفاق نمیفته چون قوانین نوع سخت‌گیرانه هستن.