شئ گرایی درc#

شئ گرایی (OOP) در C# بر چند پایه استوار است که به قرار زیرند:

Inheritance
Encapsulation
Polymorphism
Abstraction
Interface

اکنون به توضیح مختصر هر یک می پردازیم.

Inheritance

پدر و فرزندی را در نظر بگیرید. هر پدری مشخصات فردی به خصوصی دارد. فرزند وی می تواند همه خصوصیات او را به ارث برد و خصوصیتهای دیگری نیز داشته باشد که پدرش ندارد. این یعنی ارث بری! برای مثال پدر وقتی عصبانی می شود، داد و فریاد می کند. پسر هم این خصوصیت را به ارث می برد با این تفاوت که وقتی عصبانی می شود، علاوه بر داد زدن، چند عدد بشقاب هم می شکند. در برنامه نویسی شئ گرا از مفهوم ارث بری استفاده های زیادی می شود. برای تفهیم راحت تر مسئله فرض کنید کلاسی به نام وسیله نقلیه داریم .از آنجا که هر وسیله نقلیه ای حرکت می کند، رنگ دارد، سرعت دارد، ترمز می گیرد و ...می توانیم همه این متدها و فیلدها(کدام متدها و فیلدها!؟) را در کلاس وسیله نقلیه تعریف کنیم. حال یک وهله از این کلاس را در نظر بگیرید (مثلا دوچرخه!). یک دوچرخه یک وسیله نقلیه است که همه خصوصیات عمومی یک وسیله نقلیه را دارد و البته خصوصیاتی دارد که مختص خودش هستند و در انواع دیگر یافت نمی شوند. به این منظور این دوچرخه می تواند ویژگیها و متدهای مشترک را از کلاس وسیله نقلیه به ارث ببرد و در عین حال ویژگیهای منحصر به خود را نیز داشته باشد. قابلیت استفاده دوباره از کد (Reusability) یکی از مزیات اصلی ارث بری است.

Encapsulation

همانطور که از اسمش پیداست، به قرار دادن پیاده سازی در یك کپسول اشاره می کند، به طوری که کاربر بیرونی از نحوه پیاده سازی مطلع نباشد و فقط بداند که این کپسول کار خاصی را انجام می دهد. وقتی یك کپسول می خورید نمی دانید که در داخل آن چه چیزی هست و فقط به این فکر می کنید که این کپسول چه تاثیری در بدن شما می گذارد !

فرض کنید سوار ماشینی هستید که به سرعت در حرکت است! در مسیری که می روید ماشین پدر نامزدتان از روبرو به شما نزدیک می شود و سعی می کنید سریع ترمز بگیرید تا برخورد نکنید. اگر قرار باشد که بدانید بعد از فشار دادن پدال ترمز چه عملیاتی انجام می شود تا ترمز گرفته شود، دیگر باید از ازدواج قطع امید کنید. ولی اگر تنها بدانید که با فشار دادن پدال، ترمز گرفته می شود شما خوشبخت خواهید شد. در واقع ما در اینجا کار ترمز گرفتن ماشین را به صورت یک کپسول آماده در نظر می گیریم. هدف Encapsulation این است که ما را از پرداختن به ریز موضوعات رها کند و اشیاء را به صورت یک جعبه سیاهی بدانیم که به ازای یک ورودی خاص خروجی خاصی می دهند. اگر می خواهیم کدهای ما نیز این مورد را رعایت کنند باید سعی کنیم نگاه کپسولی به اشیاء و عملکرد آنها داشته باشیم. در C# برای کپسوله کردن ازAccess Modifier های protected ، privateو publicاستفاده می شود .

Polymorphism

فرض کنید پدر شما کار خاصی را به طریق خاصی انجام می دهد. مثلا برای پختن غذا (حقیقتی است تلخ!) اول

ظرفهای دیشب را شسته و بعد گاز را روشن می کند و بعد غذا می پزد! شما که خصوصیات پدر و کارهای او را به ارث می برید برای مثال برای پختن غذا ابتدا گاز را روشن می کنید، بعد کبریت می کشید، غذا را می پزید و بعد ظرفهای دیشب را می شویید! (توصیه می کنم نگذارید ظرفهایتان نشسته بمانند!) برادر شما ممکن است همین کار را به طریق دیگری انجام دهد. پختن غذا کاری است که شما از پدر خود به ارث می برید!!! ولی آن را به طریق دیگری انجام می دهید .یعنی یک کار ثابت توسط فرزندان مختلف یک پدر به طرق مختلفی انجام می شود. این دقیقا همان چیزی است که به آن چند شکلی یا Polymorphism می گویند.

Abstraction

تجرید یا مجرد سازی! به کلاسی مجرد گفته می شود که پیاده سازی متدها در آن انجام نمی شود! بر خلاف انسانها که مجرد تعریف دیگری برایشان دارد! حال سئوالی پیش می آید که اگر کلاسی داشته باشیم که نخواهیم پیاده سازی متدها را در آن انجام بدیم، از آن کلاس چه استفاده ای می کنیم؟ برای پاسخ به این سئوال شرایط زیر را در نظر بگیرید:

فرض می کنیم که شما رییس یک شرکت بزرگ برنامه نویسی هستید و می خواهید پروژه بزرگی را انجام دهید. برای اجرای پروژه از برنامه نویسان مختلفی استفاده می کنید که ممکن است همه آنها هموطن نباشند! مثلا هندی، ایرانی یا آلمانی باشند! اگر قرار باشد هر برنامه نویسی در نامگذاری متدها و کلاسهایش آزاد باشد، در کد نویسی هرج و مرج به وجود می آید. شما به عنوان مدیر پروژه، کلاسی تعریف می کنید که در آن تمام متدها با ورودی و خروجی هایشان مشخص باشند. ولی این متدها را پیاده سازی نمی کنید و کار پیاده سازی را به برنامه نویسان می دهید و از آنها می خواهید که همه کلاسهایی را که می نویسند از این کلاس شما به ارث ببرند و متدها را به طور دلخواه پیاده سازی کنند. این باعث می شود که شما با داشتن یک کلاس، ورودی و خروجی های مد نظر خود را داشته باشید و دیگر نگران برنامه نویسان نباشید. کلاسی که شما تعریف می کنید یک کلاس مجرد نامیده می شود.

برای تعریف یک کلاس مجرد از کلمه کلیدی abstract استفاده می کنیم .فیلدهایی که می خواهیم در کلاسهای مشتق شده از این کلاس پیاده سازی شوند حتما باید با abstract تعریف شوند. یک کلاس مجرد می تواند فیلدها و متدهای نامجرد داشته باشد .اگر متد نامجردی در یک کلاس مجرد تعریف کردید، حتما باید آن را پیاده سازی کنید و نمی توانید پیاده سازی آن را به کلاسهای مشتق شده بسپارید.

Interface

اینترفیس در برنامه نویسی همانند همان کلاس است تنها با این تفاوت که هیچکدام از اعضای آن پیاده سازی نمی شوند. در واقع یک اینترفیس گروهی از متدها، خصوصیات، رویدادها و Indexer ها هستند که در کنار هم جمع شده اند. اینترفیس ها را نمی توان) Instantiate وهله سازی) کرد (یعنی نمی توان وهله ای از یک اینترفیس ایجاد کرد!). تنها چیزی که یک اینترفیس دارا می باشد امضای (signature) تمامی اعضای آن می باشد. به این معنی که ورودی و خروجی متدها، نوع Property ها و... در آن تعریف می شوند ولی چیزی پیاده سازی نمی شود. اینترفیس ها سازنده و فیلد ندارند (امری است بدیهی! چرا؟). یک اینترفیس نمی تواند Operator Overloadداشته باشد و دلیل آن این است که در صورت وجود این ویژگی، احتمال بروز مشکلاتی از قبیل ناسازگاری با دیگر زبانهای .NET مانند VB.NET که از این قابلیت پشتیبانی نمی کند وجود داشت. نحوه تعریف اینترفیس بسیار شبیه تعریف کلاس است تنها با این تفاوت که در اینترفیس پیاده سازی وجود ندارد .

حالا این اینترفیس در کجا به کار می آید؟ اگر با C++ کار کرده باشید (در آن صورت کارتان خیلی درست می باشد!!!) با واژه ارث بری چند گانه آشنا هستید. ولی احتمالا شنیدید که جاوا و C# از ارث بری چندگانه پشتیبانی نمی کنند. (یعنی یک کلاس از چند کلاس دیگر به ارث ببرد .(گاهی لازم داریم از چند کلاس به ارث ببریم. راه حلش این است که از اینترفیس ها استفاده کنیم. ولی بدانید که اگر از اینترفیسی به ارث بردید باید تمام متدهای آن را پیاده سازی کنید. یک کلاس می تواند از n تا اینترفیس و تنها یک کلاس به ارث ببرد.